

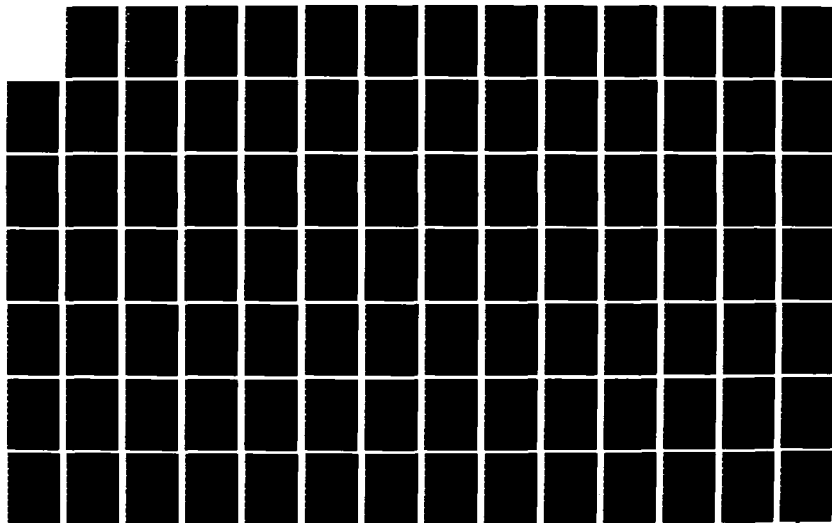
AD-A138 232

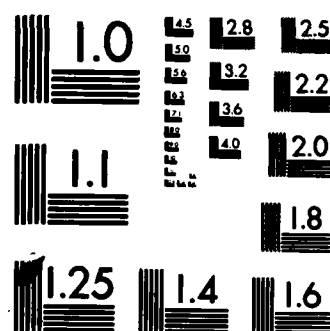
DEVELOPMENT OF A REAL-TIME GENERAL-PURPOSE DIGITAL
SIGNAL PROCESSING LABO.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. J W BENGTON
DEC 83 AFIT/GCS/EE/83D-3 F/G 9/2

1/4

UNCLASSIFIED

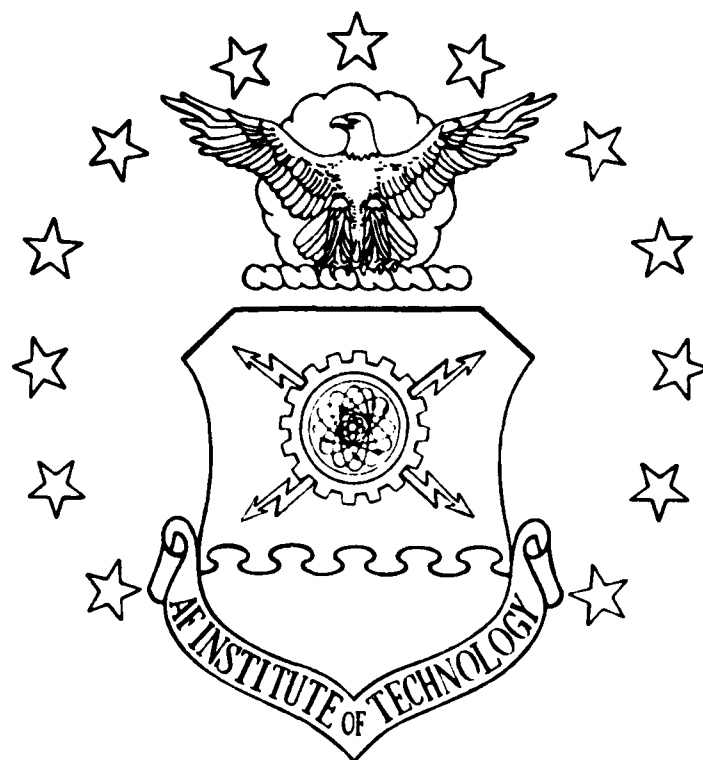
NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A138232



DEVELOPMENT OF A REAL-TIME
GENERAL-PURPOSE
DIGITAL SIGNAL PROCESSING
LABORATORY SYSTEM

THESIS

AFIT/GCS/EE/83D-3

John W. Bengtson
Capt USAF

This document has been approved
for public release and sale; its
distribution is unlimited.

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC
SELECTE
FEB 23 1984

A

84 02 22 074

DTIC FILE COPY

AFIT/GCS/EE/83D-3

DEVELOPMENT OF A REAL-TIME
GENERAL-PURPOSE
DIGITAL SIGNAL PROCESSING
LABORATORY SYSTEM

THESIS

AFIT/GCS/EE/83D-3

John W. Bengtson
Capt USAF

DTIC
SELECTED
FEB 23 1984

A

Approved for public release; distribution unlimited

Development of a Real-Time
General-Purpose Digital Signal Processing
Laboratory System

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

John W. Bengtson, B.A.

Capt

USAF

Graduate Computer Systems

December 1983

Accession For	
CHAAI	
TAB	
Unrecorded	
Location	
Distribution/	
Date	
A-1	



Approved for public release; distribution unlimited.

Preface

It's easy to make a case for the importance of Digital Signal Processing (DSP). With ever-decreasing computer hardware costs, ever-increasing gains in computer performance, and never-ending demands for improved signal processing capabilities, a wide spectrum of disciplines place heavy reliance on DSP techniques for solutions to their problems. The DSP system developed through this thesis effort is an attempt to make the benefits of DSP available to engineers with as little effort (and knowledge of computers) as possible required on their part.

Thanks to my thesis advisor, Dr. Gary Lamont, for helping things to happen. Two lessons learned at his hands warrant mention: "There's always more than one right way to do things" (an encouragement to freedom of thought and action); and "It all depends on your level of observation" (visions of endless Escher...) ϵ 's are best left as ϵ 's (the world has no lack of proper mountains).

Thanks to my mother and father for providing so much more than just my entrance to this world.

The lessons in life offered me during this year and a half bear witness to our creator's grace and succor for one "obsessed" (Ecclesiastes 12:12b comes to mind). Certainly, it is only because life is imbued with His presence that (even in the midst of AFIT!):

Life, like a dome of many-coloured glass,
Stains the white radiance of Eternity

Table of Contents

	<u>Page</u>
Preface	ii
List of Figures	vi
List of Tables	vii
Abstract	viii
Chapter 1 -- Introduction	1
1.1 -- Problem and Background	1
1.2 -- Requirements	2
1.3 -- Scope	3
1.4 -- Past Development Efforts	4
1.5 -- Approach	5
1.6 -- Results of Local User and University Survey . .	6
1.7 -- Overview of Thesis	7
Chapter 2 -- Requirements	8
2.1 -- Software Development Methodology	8
2.1.1 -- Goals for Software Methodology	8
2.1.2 -- Alternatives and Choice	9
2.2 -- User Interface	11
2.2.1 -- Rationale for a User Interface	11
2.2.2 -- On-Line Requirements	11
2.2.3 -- Off-Line Documentation	13
2.2.4 -- Definition and Importance of Real-Time Processing	13
2.3 -- Application Programs	14
2.3.1 -- Important DSP functions	14
2.3.2 -- Application Program Interaction	16
2.3.3 -- Graphics	16
2.3.4 -- Off-Line Documentation	16
2.4 -- Requirements Summary	17
Chapter 3 -- Design	18
3.1 -- Overview	19
3.2 -- Logical Design of System	21
3.2.1 -- Context DFD	23
3.2.2 -- GET_REQUEST (1.0) DFD	23
3.2.3 -- GET_OPTIONS (1.1) DFD	26
3.2.4 -- EDIT_OPTIONS (1.2) DFD	26
3.2.5 -- SATISFY_IMMEDIATE_OPTIONS (1.3) DFD	29
3.2.6 -- FORMAT_REQUEST (1.4) DFD	31
3.2.7 -- SATISFY_REQUEST (2.0) DFD	33
3.2.8 -- ACQUIRE_DATA (2.2) DFD	35
3.2.9 -- PROCESS_DATA (2.3) DFD	37

3.2.10 -- DISPLAY DATA (2.4) DFD	39
3.3 -- Physical Design of System	41
3.3.1 -- Dialogue Approach	42
3.3.2 -- Division of User Interface/Application Program Responsibilities	45
3.3.3 -- Application Program Invocation Alternatives	46
3.3.4 -- Interprocess Communication Alternatives . .	49
3.3.5 -- Physical Design Structure Charts	50
3.3.5.1 -- DSP Structure Chart	51
3.3.5.2 -- GOPT Structure Chart	53
3.3.5.3 -- DMENU Structure Chart	55
3.3.5.4 -- SIOPT Structure Chart	57
3.3.5.5 -- EOPT Structure Chart	59
3.3.5.6 -- FREQ Structure Chart	59
3.3.5.7 -- ADATA Structure Chart	63
3.3.5.8 -- PDATA Structure Chart	63
3.3.5.9 -- DDATA Structure Chart	63
3.4 -- Hardware Components	68
3.5 -- Data Acquisition Application Program Characteristics	69
3.5.1 -- AD001 (Generate Data)	70
3.5.2 -- AD002 (Sample A/D Converter)	70
3.6 -- Data Processing Application Program Characteristics	71
3.6.1 -- PD001 (Correlation and Covariance)	71
3.6.2 -- PD002 (Coherence)	74
3.6.3 -- PD003 (Convolution)	76
3.6.4 -- PD004 (FFT)	77
3.6.5 -- PD005 (FIR Filter Design)	78
3.6.6 -- PD006 (IFFT)	79
3.6.7 -- PD007 (IIR Filter Design)	80
3.6.8 -- PD008 (Waveform Averaging)	80
3.7 -- Data Display Application Program Characteristics	82
3.7.1 -- DD001 (Display to HP2648A)	82
3.7.2 -- DD002 (Display to HP1310)	82
3.8 -- General Testing Procedures and Criteria	82
3.9 -- Design Summary	86
Chapter 4 -- Implementation and Testing	88
4.1 -- Initial System Difficulties	88
4.2 -- General System Characteristics	89
4.3 -- User Interface Implementation	92
4.3.1 -- Basic User Interface Communication Modules	93
4.3.2 -- Other GREQ Subordinate Modules	96
4.3.3 -- SAREQ Module	97
4.4 -- Application Program Implementation	97
4.4.1 -- AD001 (Generate Data)	98
4.4.2 -- AD002 (Sample A/D Converter)	98
4.4.3 -- PD001 (Correlation and Covariance)	98

4.4.4 -- PD002 (Coherence)	99
4.4.5 -- PD003 (Convolution)	99
4.4.6 -- PD004 (FFT)	99
4.4.7 -- PD005 (FIR Filter Design)	100
4.4.8 -- PD006 (IFFT)	100
4.4.9 -- PD007 (IIR Filter Design)	100
4.4.10 -- PD008 (Waveform Averaging)	100
4.4.11 -- DD001 (Display to HP2648A)	100
4.4.12 -- DD002 (Display to HP1310)	101
4.5 -- Implementation and Testing Summary	101
Chapter 5 -- Conclusions and Recommendations	102
Bibliography	106
Appendix A: User's Guide	110
Appendix B: System Manager's Guide	118
Appendix C: Current and Anticipated Hardware	133
Appendix D: WPAFB HP 21MX Users	135
Appendix E: Introduction to Structured Analysis and Design	136
Appendix F: Data Dictionary (Data Items)	139
Appendix G: Data Dictionary (Program Modules)	163
Appendix H: User Interface Software Listings	191
Appendix I: Application Program Software Listings	303
Appendix J: Test Plan	326
Vita	334

List of Figures

<u>Figure</u>	<u>Page</u>
1 User's View of the System	20
2 DFD 0: Context Diagram	24
3 DFD 1: Get Request	25
4 DFD 1.1: Get Options	27
5 DFD 1.2: Edit Options	28
6 DFD 1.3: Satisfy Immediate Options	30
7 DFD 1.4: Format Request	32
8 DFD 2: Satisfy Request	34
9 DFD 2.2: Acquire Data	36
10 DFD 2.3: Process Data	38
11 DFD 2.4: Display Data	40
12 DSP Structure Chart	52
13 GOPT Structure Chart	54
14 DMENU Structure Chart	56
15 SIOPT Structure Chart	58
16 EOPT Structure Chart	60
17 FREQ Structure Chart	61
18 ADATA Structure Chart	64
19 PDATA Structure Chart	65
20 DDATA Structure Chart	66
21 Block Diagram of Correlation Method	72

List of Tables

<u>Table</u>	<u>Page</u>
I User Interface Services	12
II System Message Design Guidelines	12
III Essential DSP Functions	15
IV Dialogue Alternatives	43
V Program Invocation Alternatives	47

Abstract

This investigation resulted in the design and implementation of software to support a real-time, general-purpose digital signal processing (DSP) system. The major design aims for the system were that it: be easy to use, support a wide variety of DSP functions, and be capable of real-time processing. All work was performed using an HP21MX computer running under the RTE-III operating system.

The system's analysis and design were accomplished using Structured Analysis and Structured Design techniques. Their results -- both logical and physical system designs -- are presented via Data Flow Diagrams and Structure Charts respectively. The hardware environment was also analyzed to ensure its suitability.

The resulting system consists of two main components: a User Interface, and a collection of DSP application programs. The User Interface is menu driven and allows the system to be used by those with little or no prior computer experience. The User Interface gathers user requests and presents them to an arbitrary number of concurrently executing application programs for satisfaction. All of the major system components were successfully implemented with the exception of real-time data sampling support via analog to digital converter. With the addition of an HP21MX co-processor, the developed system should be capable of supporting the full range of DSP activities envisioned.

DEVELOPMENT OF A REAL-TIME GENERAL-PURPOSE DIGITAL SIGNAL PROCESSING LABORATORY SYSTEM

Chapter 1 -- Introduction

This chapter introduces this thesis investigation's problem, suggests its importance, broadly outlines its requirements, notes past progress, and sketches the approach taken in solving the problem.

1.1 Problem and Background

The problem this investigation deals with is the continued development of AFIT's HP21MX computer system in its role as a real-time general-purpose digital signal processing (DSP) laboratory. The system is to provide a flexible, expandable environment in which DSP engineers can work productively without needing to bother with details of the computer's operation.

To gain an appreciation for the relevance of the requirements that are discussed in the next section (and for the value of this thesis investigation), it may be helpful to consider the role played by signal processing in the world at large.

Historically, signal theory came into being with the invention of the telephone and especially of the radio. For the first time signals were transmitted and received by complex electrical apparatus, the performance of which could be subjected to mathematical analysis. Signal processing now encompasses far more than this interest in the modification

of electrical signals representing messages as they pass through electrical networks. The definition of the term "signal" now includes almost any physical variable of interest, and the techniques of signal analysis and processing are extended to areas no longer directly related to communication (Ref 28).

The ever-increasing availability of digital computing facilities has been the most important factor in the growing diversity of signal processing applications. With DSP, there is no longer a need for special-purpose electronic circuits (or even for an understanding of electrical technology) to be able to perform signal processing tasks (Ref 28:6, 36:4). Thus, DSP is both replacing other techniques in conventional applications and suggesting its own new, unique applications (Ref 47).

Given the importance of DSP, Professor Gary Lamont of the Air Force Institute of Technology has designated that the HP21MX system form the core of a DSP laboratory. In this role it should serve several important purposes: to familiarize students with the use of common DSP tools in an instructional environment, to serve as a research tool for class or thesis work, and to act as a foundation in gaining experience with developing DSP systems.

1.2 Requirements

The main requirements of the system are as follows: first, the system must be able to support the "most common"

DSP applications. Thus, the system should be designed so that any such applications not provided directly by this thesis effort are anticipated, making later expansion a well-ordered process. This requirement ensures the system's continued relevance to real-world concerns. Second, the system must be easily used by those having no experience with the HP21MX computer or with computers in general. This is important in ensuring that the system serves as a useful tool and not as a lesson in the intricacies of the computer. And finally, real-time capabilities (see Section 2.2 for a definition) and the use of graphics should be emphasized to ensure that the system is responsive and that it communicates its results in the most effective way.

1.3 Scope

This thesis investigation is directed toward the DSP-role development of the existing AFIT HP21MX system. Thus while other, newer systems may be available offering higher performance potential or complete ready-made solutions, replacement of the HP21MX system is not considered. Instead, incremental hardware and software improvements are planned (Chapter 5, Appendix C).

Making the HP21MX system into the optimal DSP system requires far more work than could reasonably be expected from a single thesis effort within the given time frame. Some work has already been accomplished in other AFIT thesis efforts (see Section 1.4). Of these, Lt. Todd's thesis (Ref

51) is the one most germane to this thesis effort. His thesis covers the broad requirements of a general-purpose DSP system (such as the type of operating system required, the various storage options available, etc.), allowing this investigation to be more narrowly focused. His results are taken as valid throughout this investigation except where noted.

1.4 Past Development Efforts

Three previous AFIT thesis investigations have worked on the development of AFIT's HP21MX system as a DSP laboratory.

The first of these was a thesis effort (Ref 48) which looked at microprogramming as a way to increase processing speed for critical sections of code. The conclusion reached was that microprogramming is useful in certain situations when the gain in efficiency justifies its coding difficulty. Another thesis effort (Ref 44) also focused on microprogramming as an approach to improving the system's processing speed, and reached the same conclusion.

Yet another thesis (Ref 50) outlined the broad requirements for a general DSP system, analyzed the HP21MX system's capabilities and deficiencies, demonstrated the system's ability to perform several basic DSP acquisition and computational tasks, and made recommendations for continued development.

Each of these thesis efforts also involved work on the HP21MX system as a whole, including the addition of new hardware elements.

1.5 Approach

These are, in order, the basic steps taken during this thesis investigation:

1. A literature search and local HP21MX user survey for information concerning: basic signal processing and DSP principles, typical DSP applications, program and system analysis and design methodologies, the HP21MX computer, available (and acquirable) DSP software, and graphics.

2. A survey of local WPAFB and AFIT HP 21MX users, Wright State instructors, University of Dayton DSP instructors, and University of Dayton Research Institute researchers for DSP software and DSP system design insight.

3. Formulation of statement of requirements for development methodology.

4. Definition of requirements for the system's user interface.

5. Definition of requirements for the system's application software.

- 6 Design, implementation, and testing of the system's user interface.

7. Design, implementation, and testing of application programs.

1.6 Results of Local User and University Survey

None of the local (WPAFB) HP21MX users have DSP software deemed useful for this system's development.

Wright State University doesn't have a DSP computer system and is doing very little DSP work. Some DSP software exists, but most of it was taken directly from Reference 13 and is located without organization on a variety of computer systems (including personal computers).

Neither the University of Dayton nor its Research Institute have a DSP computer system. While UD offered no software or DSP system design information, the Research Institute has widespread contacts with Air Force laboratories on WPAFB, some of which have DSP systems. None of those systems is based on the HP21MX so only their high-level software is transportable. Unfortunately, most of the high-level software has been written for special narrowly-focused purposes. For example, the lab in building 824 (Biodynamics and Bioengineering) has an HP21MX system used for gathering up to six channels of data from human factors test bed equipment -- but the data acquisition rate supported is quite slow (usually less than 500 Hz) and the only processing done is spectral analysis. In addition, none of the surveyed systems has a user interface of the sort envisioned for this thesis effort (user friendly), and none of them supports real-time processing with concurrent processes.

1.7 Overview of Thesis

The next chapter (Chapter 2) elaborates on the requirements for this thesis investigation. Following it are chapters on design (Chapter 3) and implementation (Chapter 4). The last chapter (Chapter 5) then presents conclusions and recommendations. These textual chapters are followed by a bibliography, various appendices, and a vita.

Chapter 2 -- Requirements

Many of this thesis investigation's requirements reflect the fact that it is primarily a software development and integration effort. While every effort will be made to borrow as much developed software as possible, all of it must be documented, modified to suit the purposes of this system, and integrated into the system's User Interface and applications software structure. The first section of this chapter considers the need to adopt a development methodology so that the software produced or adapted has certain desirable characteristics. Some of these characteristics follow directly from major goals for the system, such as ease of adding new applications. Others are taken as requirements because they have proven to be strongly related to overall system success at minimal cost (Ref 6:i). The second section specifies the requirements for the User Interface. The third section gives the application program requirements.

2.1 Software Development Methodology

2.1.1 Goals for Software Methodology

The primary goals in adopting a software methodology are to ensure that the software system is finished on time, that it satisfies all stated requirements, that it is well-documented, and that both remedial programming and the time required to later add new software components are minimized.

Completing the system on time is aided by having a well-defined approach and tool set for the entire development

process. Ensuring that the system satisfies its requirements implies a well-defined test procedure.

Minimizing the difficulty in making corrections or additions to the system has been shown to be related to certain software quality metrics, including: communicativeness, structuredness, self-descriptiveness, conciseness, legibility, and augmentability. These metrics are, in turn, part of a larger set of metrics which may be used to define other desirable aspects of software quality such as portability, reliability, and efficiency (Ref 6: xiii). A good development methodology should have some means of encouraging the presence of these characteristics. Their lack can often lead to software maintenance requiring as much as 75% of a software project's total effort (Ref 6:iii).

2.1.2 Alternatives and Choice

Some of the formal methodologies available are: the Structured Analysis Design Technique (SADT) method (Ref 5:378-409); Michael Jackson's data-structure method (Ref 5:120-175); the Warnier-Orr Structured Systems Design method (Ref 5:176-199); Yourdon and Constantine's Structured Design method (Ref 5:200-224); the METASStepwise Refinement (MSR) method (Ref 26); and the Higher Order Software (HOS) method (Ref 19). None of these methodologies is demonstrably superior to the others in general (Refs. 5:314, 5:321); in fact, new methodologies continue to be proposed as solutions to their various failings (Ref 5:297-346).

This project uses the Structured Analysis methodology presented by Victor Weinberg (Ref 53). Weinberg has borrowed analysis and design techniques from several sources (mainly Yourdon and Constantine (Ref 54)) and added his own contributions. Because of this, his method has the advantage of being a more complete "packaged" statement of methodology than many, since it deals not just with the matter of program design but with the entire development process. It borrows the concepts of data flow diagrams, data structure diagrams, and structure charts for use as tools in program analysis and design, and combines them with top-down strategies for analysis, design, implementation, and testing to form a complete method for software development from initial specification of requirements through final acceptance. Another reason for choosing Weinburg's method is its suitability for design problems in which well-defined data flows are present (input and output are clearly distinguishable from each other and transformations of data are done in incremental steps) (Ref 5:317). Well-defined data flow is typical with the type of math-intensive processing a DSP system performs. The final and perhaps most important reason for choosing Weinburg's Structured Analysis pproach is the author's familiarity with it. A brief overview of the Structured Analysis approach is given in Appendix E.

2.2 User Interface

2.2.1 Rationale for a User Interface

It has been said the the future growth of the computer industry and the acceptance of computer methods will depend largely on the successful establishment of effective man-machine communications (Ref 30:3). So also the future growth and acceptance of this system will depend in large part upon the system being not just functionally capable but also easy to use. To a large extent, the ease with which a person can communicate with a computer determines the extent to which it will be used. Making the system easy to use and efficient are the goals of a user interface, just as they are the goals of the Operating System (OS) in any computer system. While the HP 21MX's OS does take care of the four basic ease-of-use and efficiency functions (memory, processor, device, and information management (Ref 29)), it is quite complicated and difficult to learn, especially for the computer novice; the DSP User Interface remedies this. (Note that as a matter of convention, "User Interface" is capitalized when referring specifically to this system's implementation.)

2.2.2 On-Line Requirements

The User Interface complements the OS by providing the services listed in Table I for DSP system software (Ref 27).

The main emphasis here is to help the user in recovering from mistakes and in deciding what to do next. Toward those ends, one of the most important User Interface services is

Table I
User Interface Functions

1. Prompts for user input
2. Lists of options
3. Summary information for each valid command
4. User interruption of a requested action
5. Interception of user input errors
6. User allowed to correct input errors
7. Provision of default values
8. Logical sequencing of inputs
9. Directory listings

Table II
System Message Design Guidelines

System messages
should not be:

Wordy
Negative in tone
Critical of errors
General
Cryptic
Or suggest system control
over the user

System messages
should be:

Brief
Positive
Constructive
Specific
Comprehensible
And emphasize user control
over the system

Other considerations:

Upper- and lowercase letters are preferred to
uppercase only
Asterisks should be used only in special
circumstances
Error numbers, if needed at all, should appear
at the end of the message
Two or more levels of messages could be bene-
ficial
The use of terms like "illegal", "invalid", and
"error" should be avoided

the provision of on-line help: option lists, command summaries, and textual descriptions of the system's requirements and behavior at each point in the user/system dialogue. System message design guidelines to be followed by the User Interface in all of its actions are listed in Table II (Ref 45).

2.2.3 Off-Line Documentation

The concerns for the on-line User Interface may be extended to include requirements for off-line documentation. A user's guide (Appendix A) should be prepared to: present an introduction to the system, guide the user through the use of the system (some sort of tutorial), and show in detail the capabilities of each of the application programs and how they can be used with each other.

2.2.4 Definition and Importance of Real-Time Processing

One of the most-needed capabilities for the HP 21MX DSP system is real-time processing abilities. The term "real-time" has many different shadings of meaning. In this thesis investigation, it is used in two different ways. First, and in keeping with its most common definition, "real-time" denotes the system's ability to do its job quickly enough for processing results to be available within the limits imposed by time-critical operations. Second, real-time is used in describing the system's ability to perform the functions of data acquisition, processing, and result display concurrently, while allowing for user interaction to

interrupt or change the characteristics of the work being done. In general, being able to intervene while calculations are in progress, in response to the nature of intermediate results, greatly enlarges the possibilities of computation (Ref 7:11). It should be noted that AFIT's other DSP facility (the ILS system located in the Signal Processing Laboratory) does not have this second form of real-time processing. With the ILS system, only one function -- such as gathering data -- occurs at a time (Ref 50).

2.3 Application Programs

The application programs are building blocks called by the User Interface to perform work requested by the user. While never seen or directly manipulated by the normal user, their proper design and documentation is crucial to the initial and continued success of the DSP system. This section considers which DSP functions the system should be capable of performing, how they should be able to interact, and which should be implemented first.

2.3.1 Important DSP functions

Certain DSP functions stand out as being essential to a general-purpose DSP system's repertoire. These are listed alphabetically in Table III along with some of their general characteristics (Ref 1,3,13,35,38,49,50).

Table III
Essential DSP Functions

<u>Function</u>	<u>Input Domain</u>	<u>Output Domain</u>	<u>System Name</u>
Correlation and Covariance (*) Measure of internal similarity in a single signal ("auto") or between two signals ("cross")	T-R	T-R	PD001
Coherence Measure of degree of linearity in input/output behavior	T-R/C	F-R	PD002
Convolution Describe system input/output behavior in time domain	T-R	T-R	PD003
FFT (*) Time- to frequency-domain transformation	T-R	F-C	PD004
FIR Filter Design Derive FIR filter parameters	P	T,F	PD005
IFFT (*) Frequency- to time-domain transformation	F-C	T-R	PD006
IIR Filter Design Derive IIR filter parameters	P	T,F	PD007
Power Spectral Density Signal power distribution by frequency	T,F-R	F	PD002
Transfer Describe system input/output behavior in freq. domain	P	F	PD007
Waveform Average			PD008

Legend:
T=Time, F=Frequency, P=Parameters, R=Real, C=Complex
(*) = Partially implemented by Todd (Ref 37.5)

Note: the characteristics described in this table are those of the programs found in Reference 10 and may not be applicable in all cases to other programs/systems.

2.3.2 Application Program Interaction

Specifying the manner in which various programs can interact with and complement each other is a major design goal. There are two major forms of interaction which should be supported. First, when output data from one program is to serve as input data to another program, the two programs should be able to share the data directly so that the user doesn't have to manage the transfer. This applies in the case of filter design and synthesis, for example, where the data to be passed are filter parameters. The second kind of interaction involves real-time processing. The user should be able to execute a set of data acquisition, processing, and display application programs concurrently, with data passed from acquisition to processing to display automatically.

2.3.3 Graphics

Graphics software should be available to translate any large collection of data points into graphical form. This should be possible for both raw and processed data, either while it is being generated (real-time) or later from permanent storage. Three-dimensional representation is preferable in the case of complex, frequency-domain data.

2.3.4 Off-Line Documentation

A system manager's guide should be prepared, containing essential maintenance information. Its purpose is to facilitate maintenance and expansion of the DSP system by describing: its overall structure, the structure,

capabilities and limitations of each of its application programs, hardware characteristics (especially such things as D/A converters), and aids to help in using the HP 21MX and its operating system.

2.4 Requirements Summary

All of this thesis effort's essential requirements were presented in this chapter. They included: the selection of a development methodology (Section 2.1, Weinberg's Structured Analysis); the goals and characteristics of the system's User Interface (Section 2.2, notably ease of use); and application program characteristics (Section 2.3, Table III).

This thesis contains no separate chapter devoted to the analysis stage of system development. One reason for this is the fact that this is not an effort to automate or extend an existing, operational DSP system. Modeling of the "current system" is one of the primary goals of the analysis stage, and does not apply in this case. A second reason is that, at least from the user's view, the system to be developed is rather straightforward -- the system's design follows quite readily from its requirements. Thus, all analysis results and other support for the system's design are naturally and unobtrusively included in the text of the design chapter (Chapter 3).

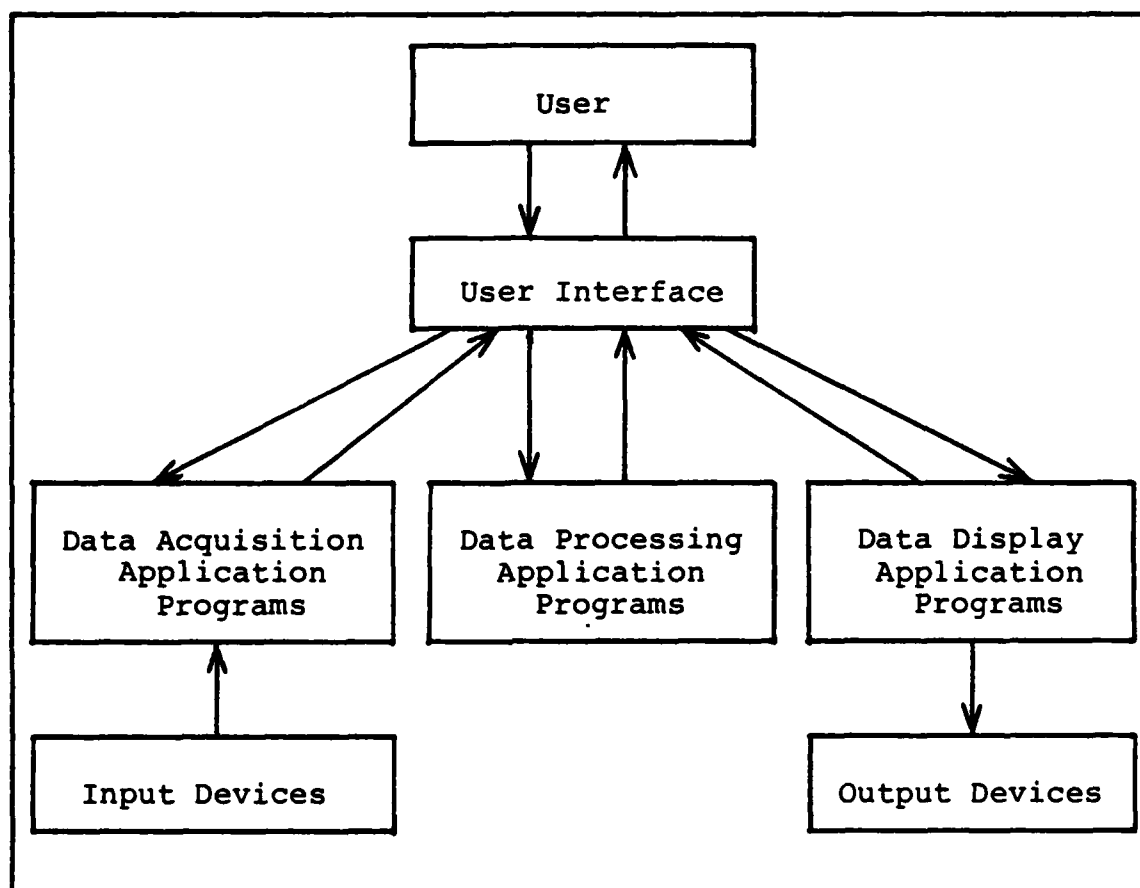
Chapter 3 now presents the system design developed to satisfy the requirements of this chapter.

Chapter 3 -- Design

This chapter presents first an overview of the system's operation, then the logical design of the system, then its physical design, and finally specific characteristics of the system's hardware and software. The overview section (Section 3.1) gives a picture of how the system will appear to a user. The logical design section (Section 3.2) describes the system's basic capabilities and its global design characteristics. The physical design section (Section 3.3) provides a more detailed view of the system's structure and of its control and coordination characteristics. The hardware components section (Section 3.4) describes the hardware that is currently available on the HP21MX DSP system. The data acquisition program section (Section 3.5) treats those programs responsible for acquiring the system's data. Section 3.6 describes the system's data processing application programs which perform: correlation, covariance, coherence, convolution, FFT, FIR filter design, IFFT, IIR filter design, and waveform averaging functions. The data display program section (Section 3.7) characterizes the system's data display programs, which transform data into graphic form. The test section (Section 3.8) lists the test procedures to be applied to all system programs during implementation.

3.1 -- Overview

Figure 1 gives a "user's view" of the HP21MX DSP system. The user communicates through the User Interface. The User Interface serves to coordinate and support (with help text, for instance) the activities of the programs responsible for the system's real DSP work -- the data acquisition, processing and display programs. The User Interface prompts for, collects, edits, organizes, and distributes user requests to the application programs. The application programs then acquire, process, and display data according to the user's requests. They are able to communicate both with one another and with the User Interface, so that a number of them may be executing at the same time, passing data through what might be termed a data pipeline. The user is in control of the system at all times, able to start, stop, and alter the system's activities at will.



3.2 -- Logical Design of the System

The purpose of a logical design is to show the flow of data through a system's components without reference to low-level physical details such as devices, storage media, variable names, and specific file names. The logical design presented in this section is the first attempt to give shape to a system reflecting Chapter 2's requirements. Data Flow Diagrams (DFD's) are used here to specify the system's logical design (DFD's were chosen as the tool for this task in Subsection 2.1.2). The following subsections describe their respective DFD's. Again, there are many additional details given in the physical design section (Section 3.3). The DFD names and their associated diagram and figure numbers are:

<u>Diagram</u> <u>Number</u>	<u>Figure</u> <u>Number</u>	<u>Name</u>
0	2	Context DFD
1.0	3	GET_REQUEST
1.1	4	GET_OPTIONS
1.2	5	EDIT_OPTIONS
1.3	6	SATISFY_IMMEDIATE_OPTIONS
1.4	7	FORMAT_REQUEST
2.0	8	SATISFY_REQUEST
2.2	9	ACQUIRE_DATA
2.3	10	PROCESS_DATA
2.4	11	DISPLAY_DATA

As an example in interpreting DFD's, consider Figure 3 (Get Request, Diagram 1.0). Each circle in a DFD represents a transformation to be applied to data. A transform may be some sort of mathematical operation, an edit operation, etc. In the case of Diagram 1.0, there are four such transforms:

"Get Options", "Edit Options", "Satisfy Immediate Options", and "Format Request". The transforms on each diagram are consecutively numbered starting at 1. The data provided to the transforms are represented by labelled arrows such as "Options" (input to Transform 1). Likewise, data which is being output from a transform following its transformation is represented by a labelled arrow. "Edited Options " is thus both an output from Transform 2 and an input to Transform 4. Boxes represent sources or sinks of data. Here, the user of the system is a sink for data -- receiving the system's response to immediate option requests from Transform 3. Horizontal lines are data files (repositories for data of some sort, not necessarily disk files) with their names listed underneath. "Menu File" is Diagram 1.0 data file. Each of the transforms in a DFD may in turn be further broken down by a subordinate DFD which further details the sequence of transforming operations it performs. Each DFD is identified by concatenating the transform numbers of all of its superordinates, in order, from the topmost diagram (Diagram 0) on down. Thus Transform 2 ("Edit Options") of Diagram 1.0 is further broken down in Diagram 1.2, and so on. Transforms that are not broken down further in subordinate DFD's have an asterisk inside the circle beneath their name (neither of the transforms of Figure 4, for example, are broken down further).

3.2.1 -- Context (0) DFD

This DFD (Figure 2) shows the DSP system in its simplest, highest-level form. The first data to enter the system are user requests for services, with the user being prompted as required. Transform 1 (GET_REQUEST) is responsible for handling all interaction with the user, which includes mostly accepting and editing requests and providing help. Transform 2 (SATISFY_REQUEST) represents the collection of application programs available to handle signal acquisition, processing, and display needs.

Separating the system into these two major pieces is a natural way of functionally decomposing the work to be accomplished; the system finds out what's expected of it (GET_REQUEST), and then does it (SATISFY_REQUEST).

3.2.2 -- GET_REQUEST (1.0) DFD

Within the GET_REQUEST DFD (Figure 3), GET_OPTIONS (Transform 1.1) initiates and continues a dialogue with the user by prompting for user options. The user's options are normally passed to EDIT_OPTIONS (Transform 1.2) where they must satisfy certain criteria before being accepted and acted upon. If the options selected by the user don't involve any DSP processing per se, such as requests for help, then the options are passed to SATISFY_IMMEDIATE_OPTIONS (Transform 1.3) and satisfied. Accepted non-immediate requests, such as requests for data acquisition and processing, are finally passed from EDIT_OPTIONS to FORMAT_REQUEST (Transform 4)

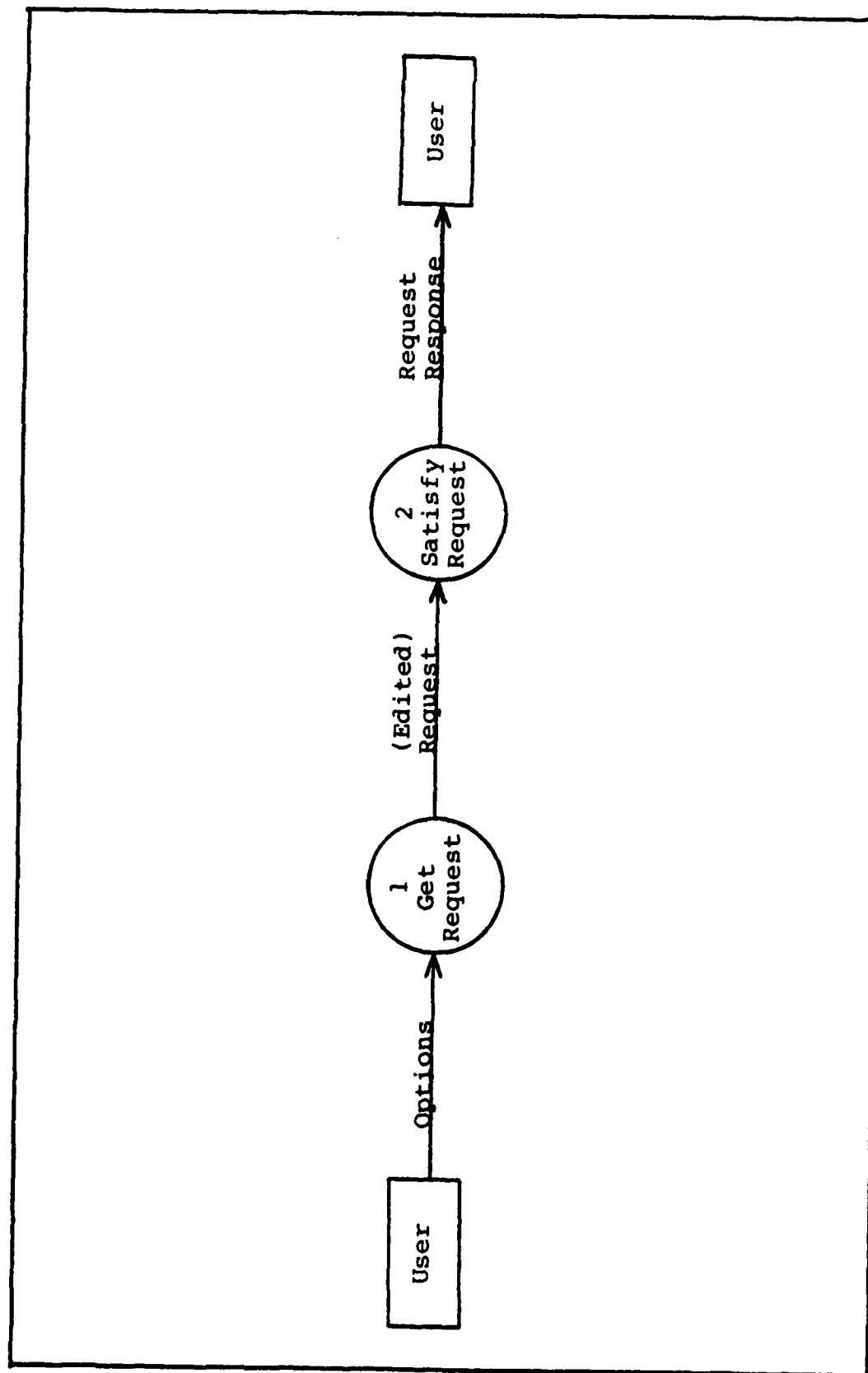


Figure 2
Diagram 0: Context Diagram

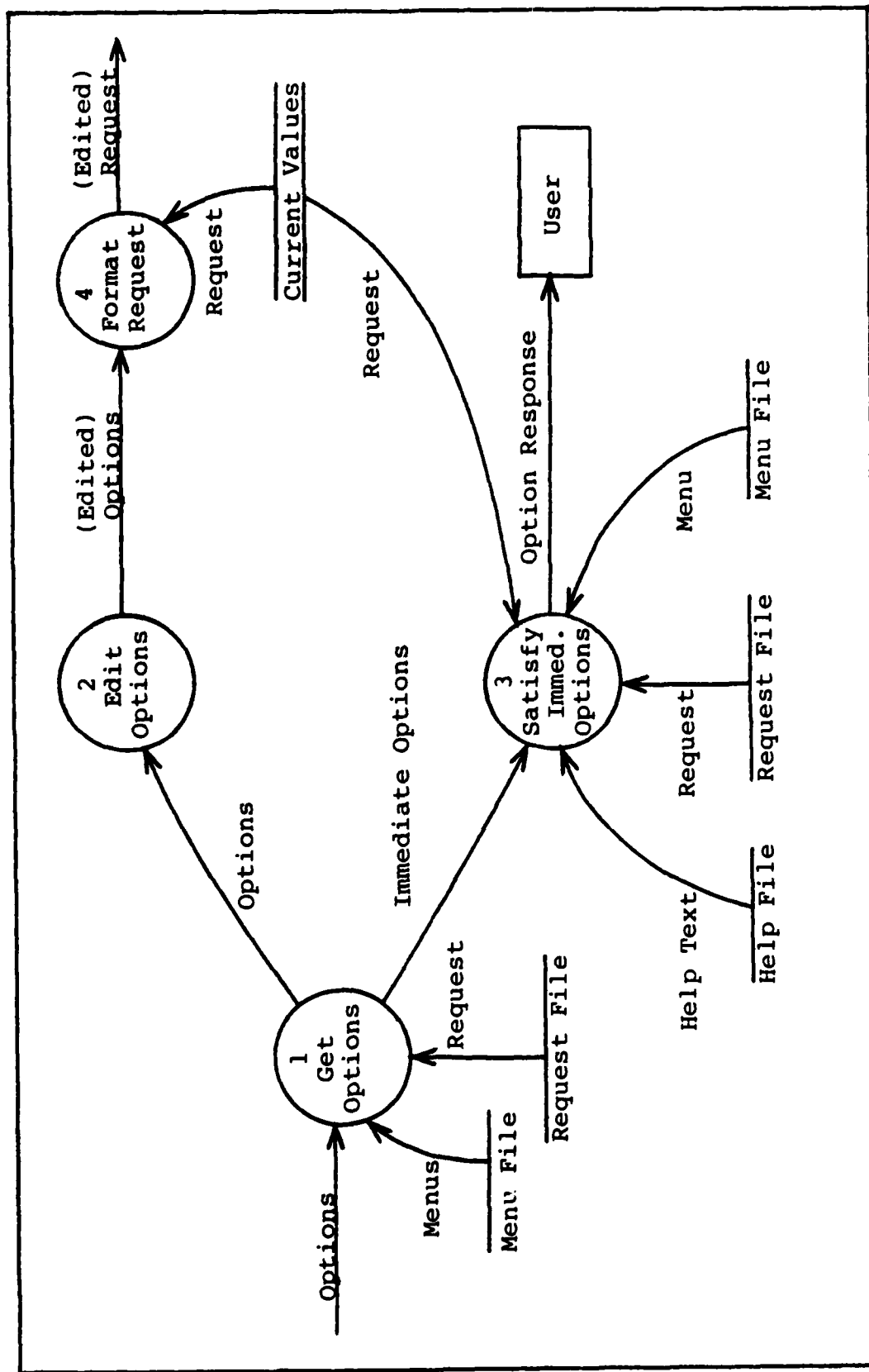


Figure 3
Diagram 1: Get Request

where they are transformed into the form expected by the application programs to which they apply.

3.2.3 -- GET_OPTIONS (1.1) DFD

GET_OPTIONS (Transform 1.1, Figure 4) is responsible for handling all interaction with the user. The user is prompted for input by DISPLAY_MENU (Transform 1.1.1). The prompts include (as per system requirements stated in Chapter 2) default values for all requested information. The user's responses are accepted by ACCEPT_OPTIONS (Transform 1.1.2) and then passed on to be edited by EDIT_OPTIONS (Transform 1.2).

3.2.4 -- EDIT_OPTIONS (1.2) DFD

The transforms that make up EDIT_OPTIONS (Transform 1.2, Figure 5) are responsible for identifying any unacceptable user options and for providing appropriate error messages. DISTRIBUTE_EDIT_OPTIONS (Transform 1.2.1) distributes the user's options to a particular EDIT_MENU transform (Transforms 1.2.2-1.2.N), which is then responsible for confirming the proper content of that set of options. The notion of a screen menu is admittedly a physical detail (treated in section 3.3), but is presaged here for the sake of continuity. Options which are accepted by their editing module are sent on their way to be made a part of a set of complete user requests for execution later at the user's bidding.

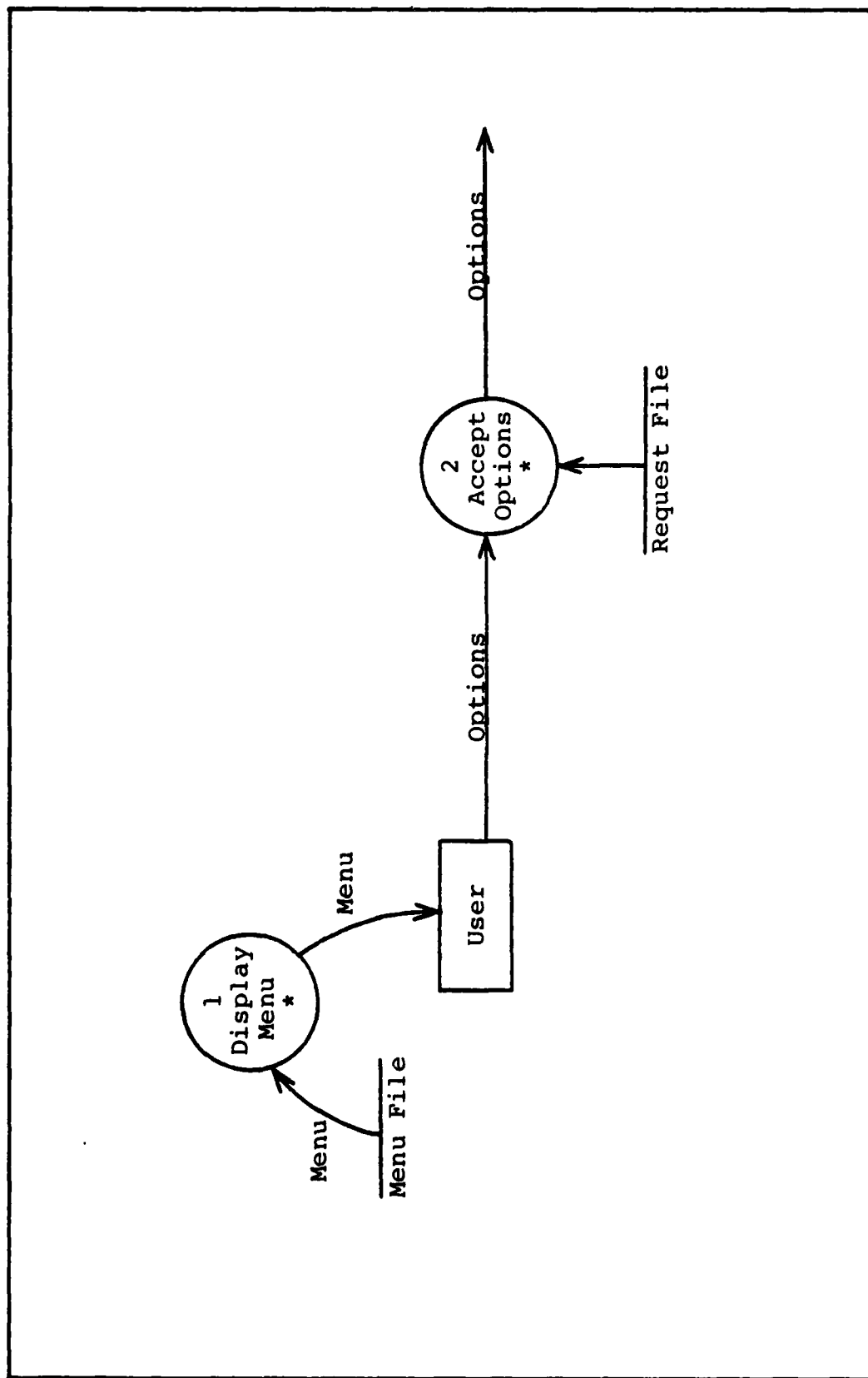


Figure 4
Diagram 1.1: Get Options

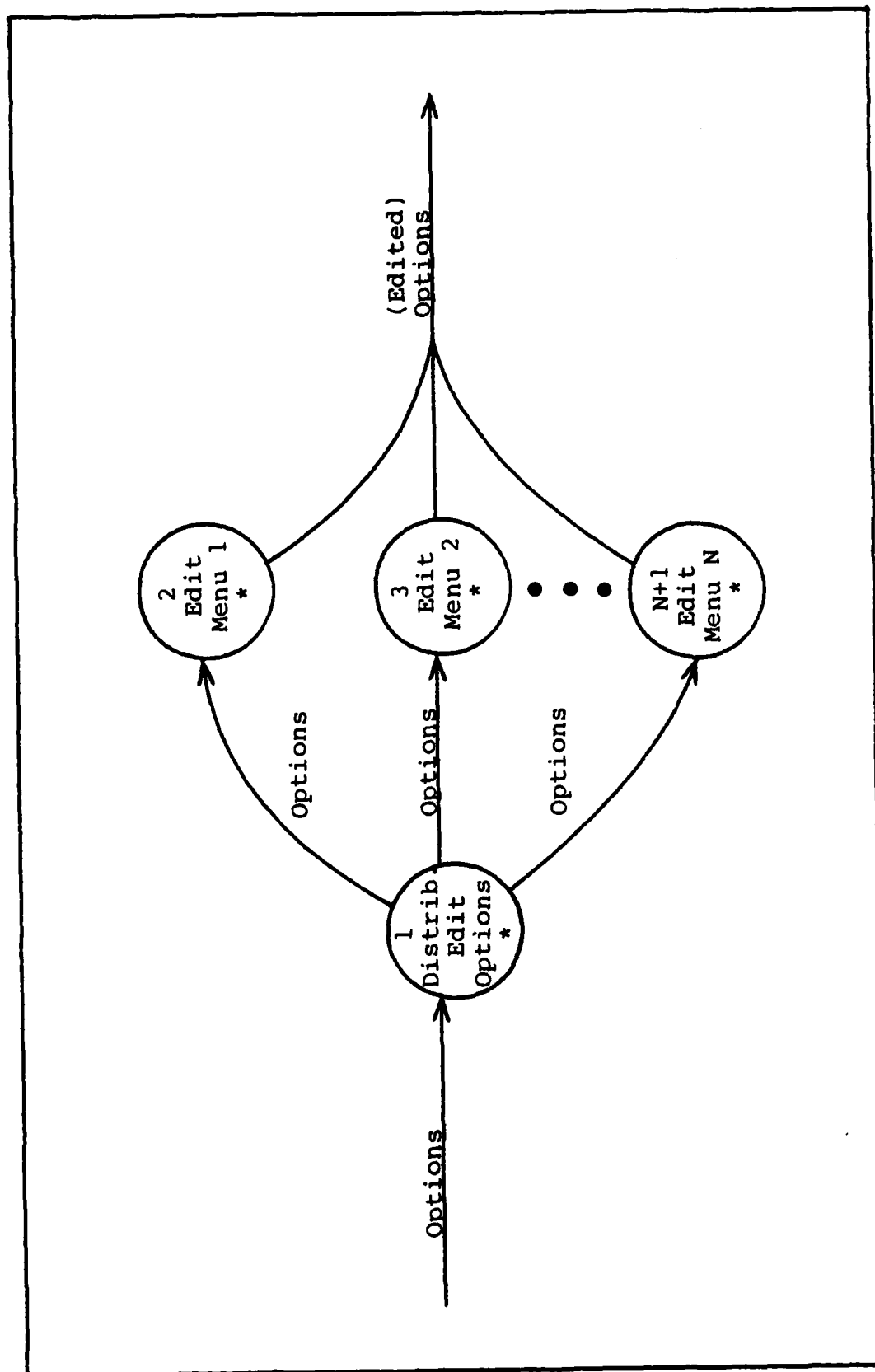


Figure 5
Diagram 1.2: Edit Options

3.2.5 -- SATISFY_IMMEDIATE_OPTIONS (1.3) DFD

SATISFY_IMMEDIATE_OPTIONS (Transform 1.3, Figure 6) takes care of all user services which aren't really a part of an actual DSP work request. DISTRIBUTE_IMMEDIATE_OPTIONS (Transform 1.3.1) gives the user-specified option to one of Transforms 1.3.2-1.3.7 for action. The first of these, PROVIDE_HELP (Transform 1.3.2), reads a help file containing text directed toward the current menu which should help the user in interpreting prompts and understanding more fully the consequences and limitations of available options. PROVIDE_DIRECTORY (Transform 1.3.3) provides a directory listing of disk files in case the user has forgotten the name of one needed for input or wants to ensure that the name chosen for a new file isn't a duplicate of one that already exists. The operation of TRAVERSE_MENU_TREE (Transform 1.3.4) is transparent to the user. Its function is to coordinate the order in which menus are presented to the user, taking care of both normal sequencing and express user requests to deviate from the normal flow of data entry (such as requests to exit the system or to back up to the previous menu). SAVE_REQUEST (Transform 1.3.5) gathers all of the user's options (which cumulatively form a "request") and saves them on a file for later recall and use. READ_REQUEST (Transform 1.3.6) reads in a previously stored request, treating it as current terminal input from the user. SAVE_DEFAULTS (Transform 1.3.7) takes the values which were provided by the user for the current menu (along with any

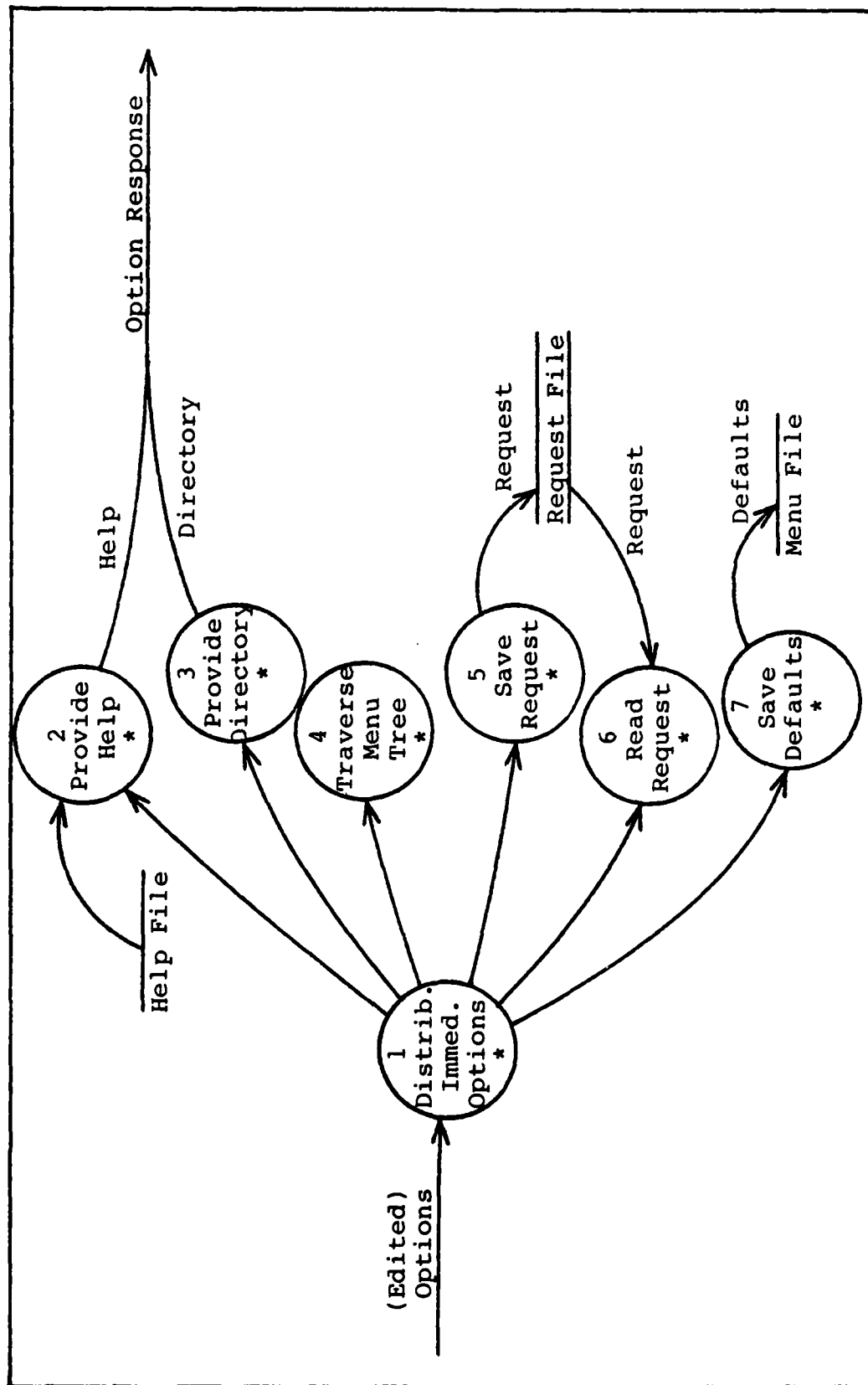


Figure 6
Diagram 1.3: Satisfy Immediate Options

default values which were left unchanged) and saves them in the menu file for the current menu, so that they will appear as the menu's default values until changed again. Note that SAVE_REQUEST doesn't affect default values; it simply allows a user request to be saved so that when it is later read in, the values it contains are substituted for the default values just as though the user had entered them from the terminal. Similarly, READ_REQUEST doesn't affect default values except to replace certain of them for the current terminal session.

3.2.6 -- FORMAT_REQUEST (1.4) DFD

The FORMAT_REQUEST transform (Transform 1.4, Figure 7) is needed to translate user requests from the form in which they're stored by the user interface into the form expected by the system's application programs. This is necessary because (speaking physically again) the application programs are not required to be consistent in their description and use of data items. For example, the order of entry or required type of data values (e.g., REAL vs. COMPLEX) might vary from one FFT to another. This allowance for data structure variance from one application program to the next means that programs may be taken from diverse sources and integrated into this system without having to rewrite them. Rather than having each module of the user interface keep track of the many different forms their data must be in to satisfy various application program requirements (a requirement which could mean significant reworking of modules

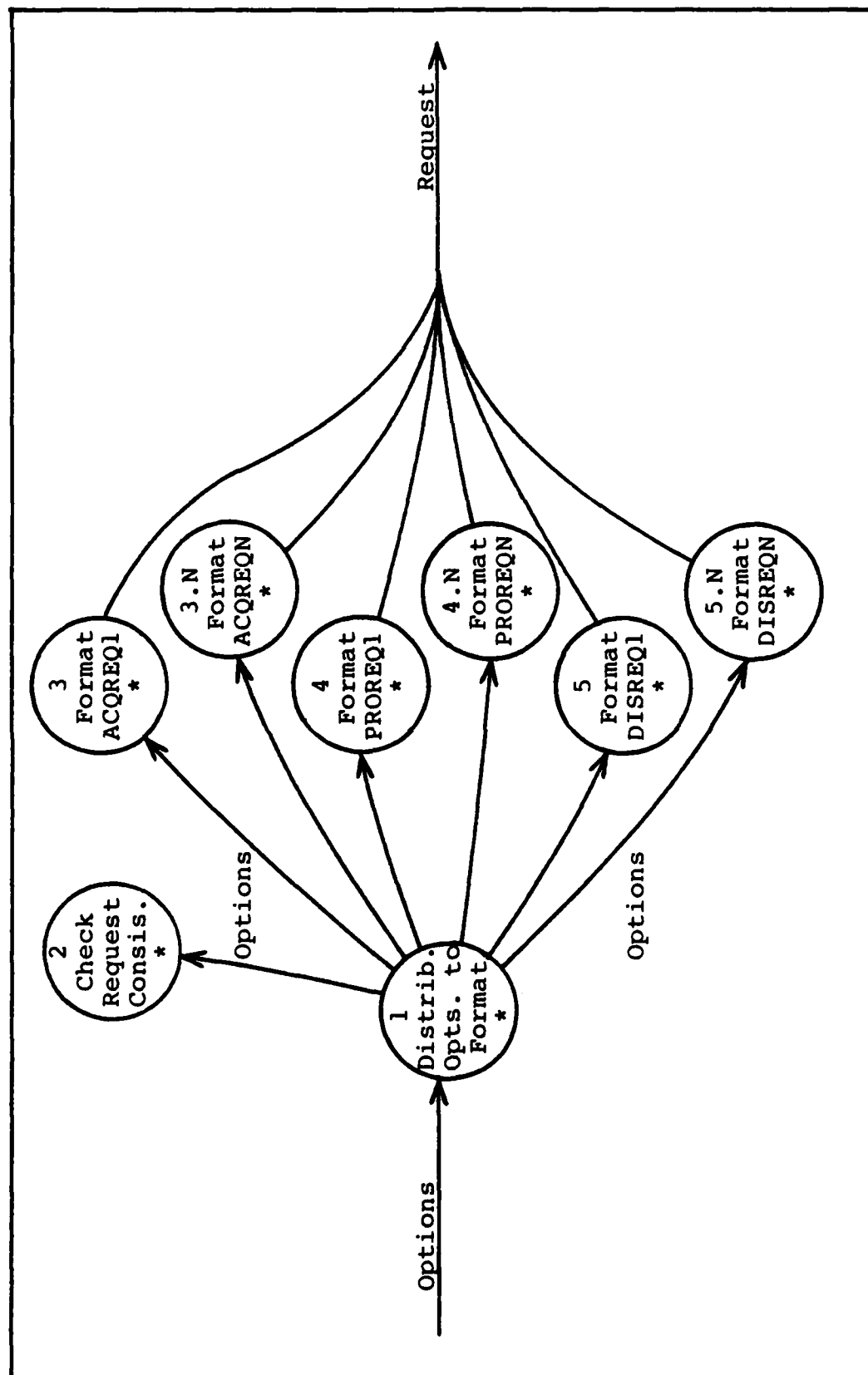


Figure 7
Diagram 1.4: Format Request

throughout the system with each newly added application program), consistent data structures are used throughout the user interface and translated to satisfy application program needs by the format transforms. Note that there is one formatter module for each application program.

CHECK_REQUEST_FOR_CONSISTENCY (Transform 1.4.2, still Figure 7) ensures that requests being sent to different application programs are consistent. For example, if an acquisition program is told to provide 512 sample points, and the follow-on processing program is told to process 1024 points, an incompatibility may exist and, if so, must be corrected before the user's request can be executed. Error messages (and help if requested) are provided to assist the user in fixing any problems.

3.2.7 -- SATISFY_REQUEST (2.0) DFD

Once a set of user requests has been formed, they are submitted to SATISFY_REQUEST (Transform 2, Figure 8). DISTRIBUTE_REQUEST (Transform 2.1) is then responsible for parceling out the requests according to whether they are requests for data acquisition, processing, or display. ACQUIRE_DATA (Transform 2.2) and DISPLAY_DATA (Transform 2.4) will never have more than one request directed to them at a time, and in fact may not have any directed to them at all during a given system run. Thus, the system gathers data from only one source at a time, and provides a graphical display of its results to only one destination at a time.

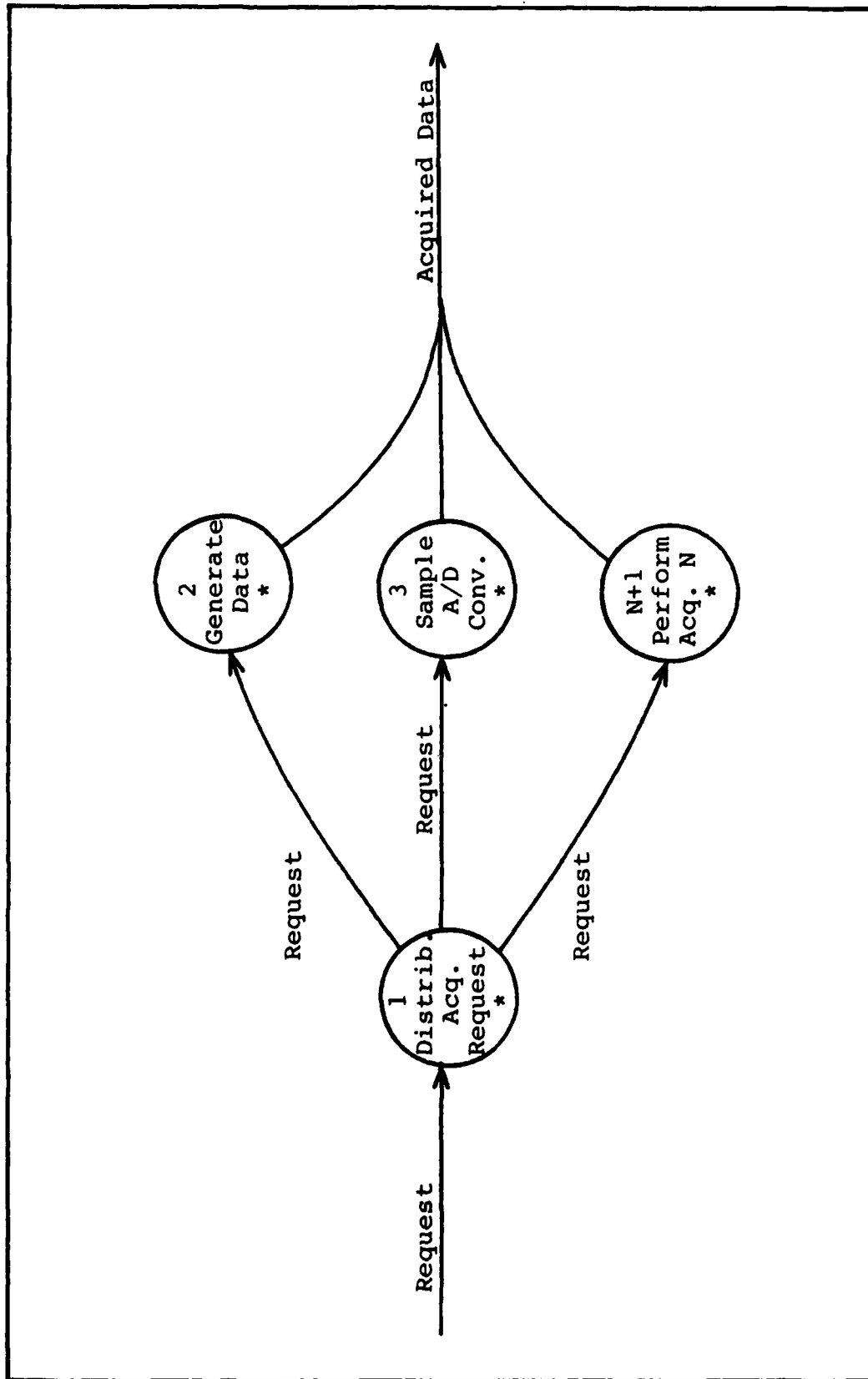


Figure 9
Diagram 2.2: Acquire Data

takes values (samples) from an analog-to-digital converter that's hooked up to some real-world signal of interest. Both GENERATE_DATA and SAMPLE_A/D_CONVERTER can either pass their data on to a processing transform (application program) or store it in a file for later use.

3.2.9 -- PROCESS_DATA (2.3) DFD

The PROCESS_DATA transform (no. 2.3, Figure 10) performs all of the system's number-crunching, including FFTs, filter design, etc. DISTRIBUTE_PROCESSING_REQUEST (Transform 2.3.1) is responsible for distributing processing requests to their appropriate processing transforms (application programs) and for setting up controls so that multiple application programs executing at the same time cooperate properly. This includes such things as the passing of data from one process to the next, and termination of all processes when a fatal error occurs. Note that each transform may, in general, communicate in a number of ways. Input data to be processed may come from another transform (either acquisition or processing), from a raw (acquired) data file prepared earlier by one of the acquisition transforms, or from a processed data file prepared earlier by a processing transform. Output data may be sent to another transform (either processing or display) or to a processed data file. In addition to normal input and output data transfers, parameter transfers are also supported. Thus, a transform whose output is a set of parameters which may be used to control another transform's

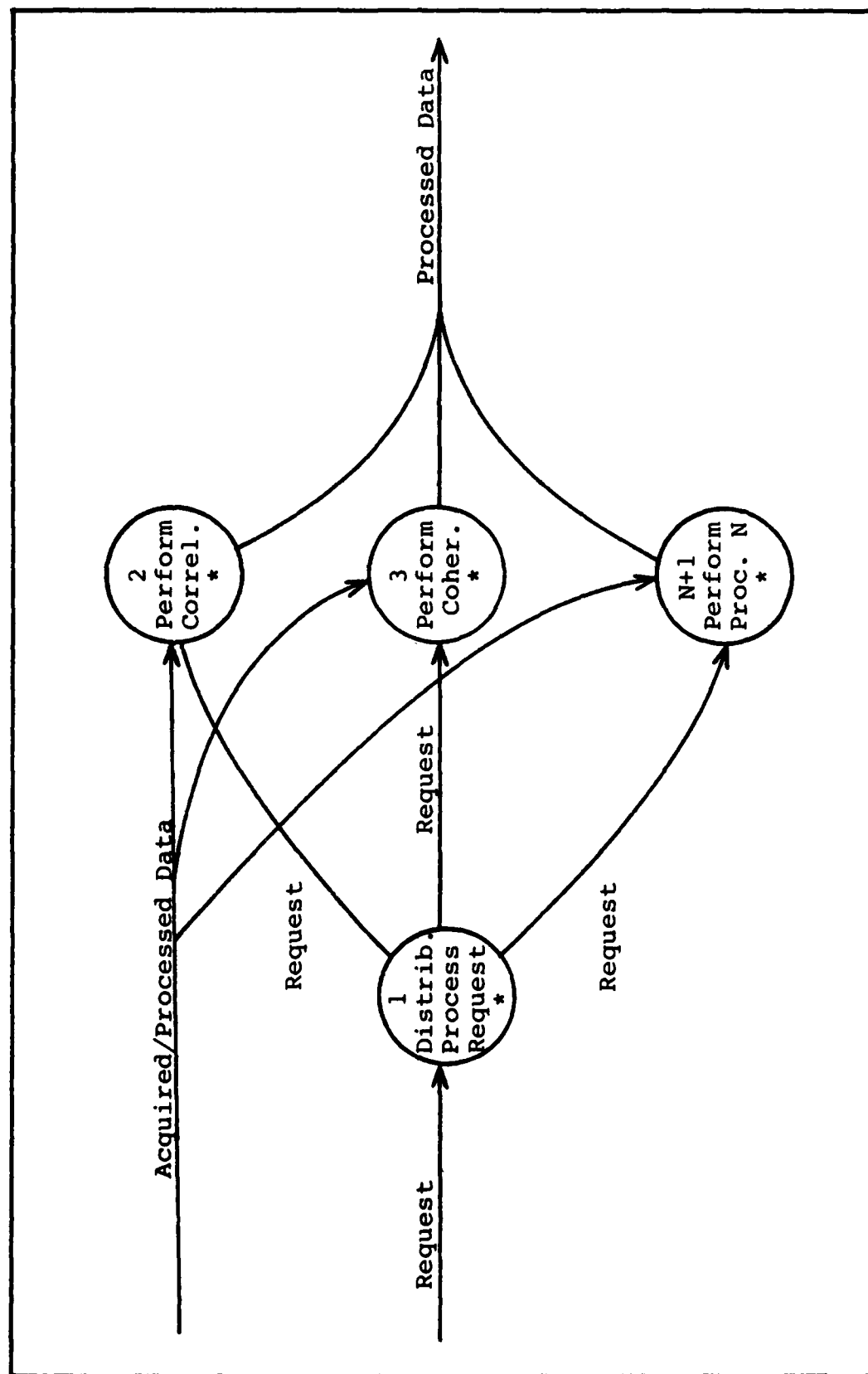


Figure 10
Diagram 2.3: Process Data

actions on actual data is able to pass those parameters directly to the using transform through a data file. An example of this is a filter design program passing its output (the filter's defining parameters) to a filtering program.

3.2.10 -- DISPLAY_DATA (2.4) DFD

DISPLAY_DATA (Transform 2.4, Figure 11) takes acquired or processed data and transforms it to graphical form before presenting it to the user on some output device. DISTRIBUTE_DISPLAY_REQUEST (Transform 2.4.1) sends the request to the appropriate display module (of which there may be arbitrarily many). DISPLAY_TO_HP2648A (Transform 2.4.2) handles output which is to be presented to the HP2648 terminal. DISPLAY_TO_HP1310 (Transform 2.4.3) is responsible for output to the HP1310 graphics terminal.

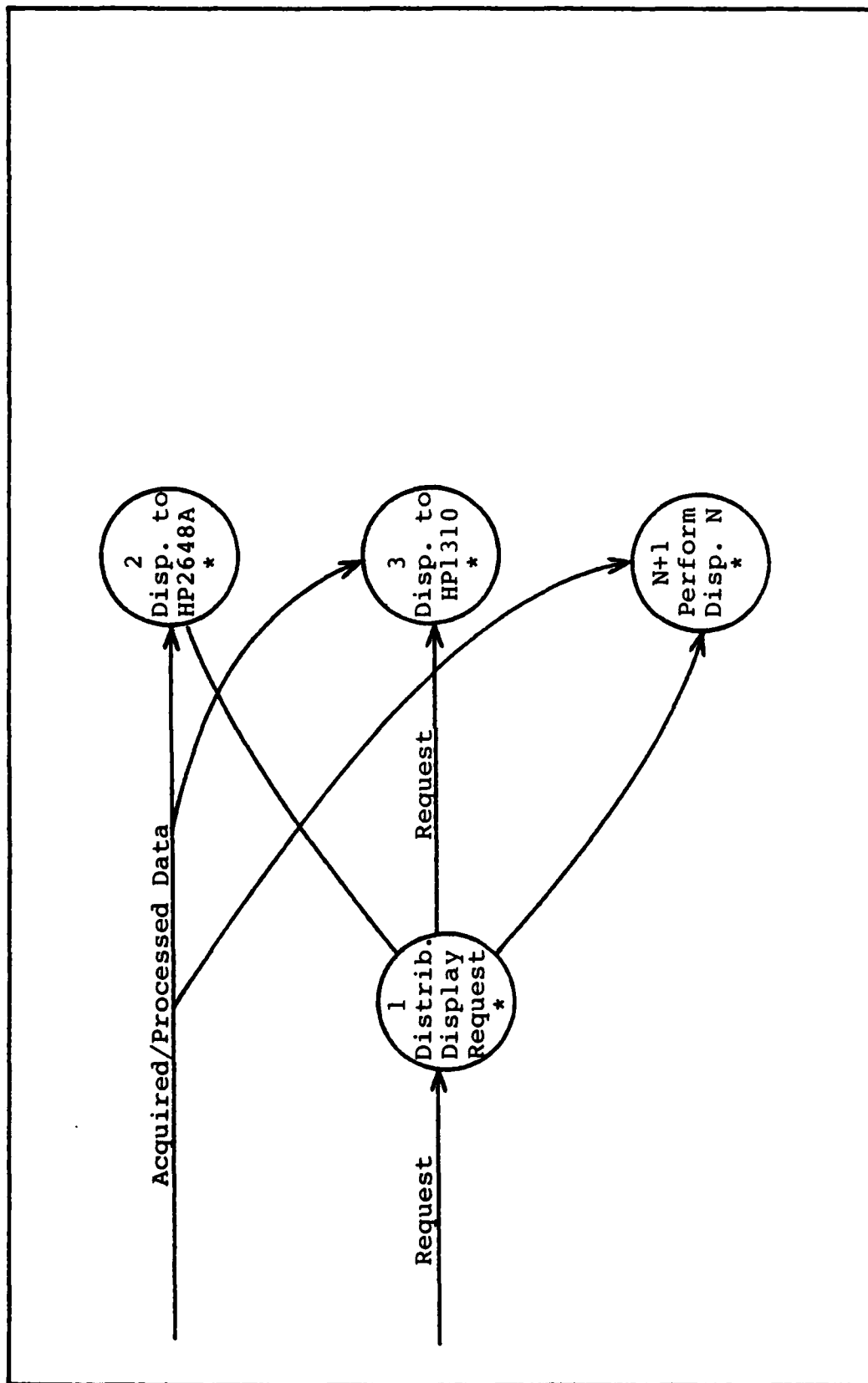


Figure 11
Diagram 2.4: Display Data

3.3 -- Physical Design of System

Developing a physical design is mainly a process of translating the logical design into a form which more closely resembles what the finished software product will look like, filling in a lot of detail along the way. In general, each of the completely decomposed transforms specified in the logical design represents a software module in the physical design. Certain modules appear in the physical design that were not a part of the logical design. This is due to the fact that some modules have fairly physical functions whose need could not be foreseen at the logical level (even more of them are added during system implementation for the same reason).

Structure Charts are used in this section to pictorially describe the system's physical design in the same way that Data Flow Diagrams were used to describe the system's logical design in the last section. In fact, techniques known as Transform Analysis and Transaction Analysis (Ref 53:176-187) were used to create most of the Structure Charts from the Data Flow Diagrams (Ref 53). There are other ways to handle the transformation of DFD's to Structure Charts, such as direct linear mappings, but Transform and Transaction Analysis are the only techniques widely cited and recommended for use with the Structured Analysis Methodology (Ref 53,54). Subsection 3.2.5 contains the Structure Charts along with their descriptions. The data item and module names used in the Structure Charts are defined (and cross-referenced to

logical design variants) in Data Dictionaries (Appendices F and G).

Before presenting the Structure Charts, several design issues are dealt with in subsections 3.2.1-3.2.4.

3.3.1 -- Dialogue Approach

Dialogue with a computer can take many forms, each form having its own advantages and disadvantages. The most important concern affecting the choice of this system's dialogue form is that it be easy to use. Several alternatives are listed in Table IV, along with their advantages and disadvantages (Ref 30).

Among the approaches considered, several had major shortcomings which led to their being discarded. A natural-language dialogue would appear to be the best form of user-interface communication. Unfortunately, such an approach is extremely difficult to implement. Dialogues using mnemonics are used in other DSP systems (such as the Air Force Institute of Technology's ILS system), but require the operator to be familiar with the mnemonics and their formats. A dialogue using programming-like statements has merit in its precision and conciseness, but requires a well-trained operator with some programming aptitude. The use of programming languages for user-interface communication requires a user well-versed in programming, which violates the requirement that this system be usable by those having little or no computer experience.

Table IV
Dialogue Alternatives

Natural-language dialogue

Advantages: theoretically the most natural man-machine interface.

Disadvantages: unsuitable where an operator's input must be interpreted with precision because of the ambiguity of language. Immense software problems.

Dialogue using mnemonics

Advantages: can be concise and precise.

Disadvantages: operator must be familiar with mnemonics and formats.

Dialogue with programming-like statements

Advantages: can be concise and precise.

Disadvantages: operator must be well-trained, familiar with the coding, and have limited programming aptitude.

Programming languages

Advantages: concise, precise, powerful, flexible.

Disadvantages: inappropriate for vast majority of computer users (non-programmers).

Question and answer dialogues (computer asks operator a series of questions)

Advantages: very simple for the operator. Simple to program.

Disadvantages: limited flexibility.

Computer-initiated dialogue

Advantages: can be used with a totally untrained operator.

Disadvantages: dialogue can be lengthy and slow. Little flexibility in sequence of operation.

Form-filling dialogue

Advantages: straightforward for operator except for cursor manipulation.

Disadvantages: less flexible than a "branching tree" of questions, and error correction procedures less easy.

Menu-selection dialogue

Advantages: simple for the operator. Can be written with a simple program generator.

Disadvantages: limited in scope.

The remaining forms of dialogue (question and answer, computer-initiated, form-filling, and menu-selection) all have simplicity of operation as one of their advantages. Their common disadvantage is limited flexibility. Unfortunately, there is an inherent tradeoff between novice ease of use and flexibility which can't be circumvented. For example, if the user is to be prompted by menus for all input, he is necessarily constrained to entering data in the order and format dictated by those menus. While some flexibility should be permitted in the order of data entry, too much flexibility would present the user with many more decisions and ambiguous situations than would a tightly controlled data entry sequence. The developed system will represent an attempt at achieving a reasonable balance between ease of use and flexibility.

This system makes use of each of the four easy-to-use types of dialogue. The question and answer and computer initiated approaches free the user from having to know what actions are expected -- the user is prompted for input. Each dialogue interchange with the user is physically centered around a screen form. This form filling allows data on the screen to be protected, so that the user doesn't accidentally erase or write over prompting messages. It also provides a means of pre-editing, by physically preventing the user from entering data values containing too many characters. Finally, each of the forms is presented in a consistent menu selection format so that the user is always shown all

available options and may always use the same procedures for responding, no matter what the inquiry.

3.3.2 -- Division of User Interface and Application Program Responsibilities

It may not be obvious why the user interface is given the tasks of prompting for input, correcting errors (even those at levels intimately associated with the operation of specific application programs), and providing default values. Admittedly, this requires extra communication between the user interface and application programs. Why not localize these activities, making each application program responsible for its own prompting, etc.? One reason for centralizing rather than localizing is to keep from having to duplicate the menu generation code involved in prompting. Also, error-handling conventions are more easily and consistently implemented when error-handling code is kept together rather than being divided among the modules served. Another reason is to decrease the need to tamper with already-developed application programs that are being incorporated into the system, so that errors are less likely to be introduced into them. Finally, centralizing all error correction, etc. keeps application program size to a minimum, allowing all available memory space to be dedicated to essential program logic and data storage (this is important because the memory available per contiguous program is quite limited -- 17K words at present (Ref 39:6.14)).

3.3.3 -- Application Program Invocation Alternatives

Once the user interface has gathered the user's requests, it must arrange for various application programs to carry out those requests. Table V presents three different methods by which one program may invoke others on the HP21MX. The first method, linking programs together into a single program unit, may be dismissed out of hand because it doesn't allow arbitrarily large sets of code to be executed (within the limits of physical computer memory). Such a restriction would severely limit the system's flexibility in combining programs for real-time processing. The second method is program segmentation (Ref 39:4.47). Note that HP uses the term "segmentation", which usually refers to a sophisticated memory management scheme (Ref 29:165-181), instead of "swap", which is really all that's being done (a program or subroutine is called by and replaces one in memory). The HP's segmentation scheme may be viewed as a crude form of overlaying (Ref 29:186) -- crude because all partitions are fixed in size, all overlaying is explicit controlled by calls within the programs, and because the overlaying provides a transfer of control in one direction only (the overlayed/called program must call back its caller to return control to it, and control passes to the first executable statement in the caller rather than to the point from which it called the overlay). Segmentation may be dismissed because it is very slow without offering any offsetting advantages. For maximum flexibility and nearly optimal speed

Table V
Program Invocation Alternatives

1. Linked Programs

With the linked program method, all programs that are needed to satisfy some user request (a program set) are linked together so that they appear as a single program unit.

Advantages:

Fast executing method, since programs all reside in memory together (no swapping or loading from disk).

Disadvantages:

Program set size severely limited (17K words under present operating system, RTE-III).

Every possible combination of acquisition, processing and display programs would have to be linked & stored in a file. Thus a change to one program might mean having to re-compile and re-store a great many program sets.

2. Segmentation (Ref 29:4.47)

With segmentation, each program that is invoked (brought into memory and executed) is loaded into memory on top of the invoking program. This technique is also known as overlaying.

Advantages:

No limit on the amount of code which may be executed (arbitrarily many program segments may be successively invoked).

(Continued)

Table V (Continued)
Program Invocation Alternatives

Disadvantages:

Programs must be loaded from disk before being executed each time they are invoked; thus, the process is very slow.

Only one program is executing at a time, so system resources aren't used efficiently (e.g. data acquisition and processing could be occurring simultaneously), which bodes ill for real-time processing.

CLASS I/O communication may not be used for transferring data from one segment to the next and for coordination. Only FORTRAN COMMON is permitted, the use of which is bad programming practice in general.

3. Concurrent Programs (Ref 29:4.49)

The HP21MX's operating system (RTE-III) allow several programs to be executing at the same time, subject only to memory partition availability. One program may start another one executing (while it continues its own execution) by using EXEC calls (Ref 29:3.24).

Advantages:

Program set size limited only by available memory (segmentation could be used on any programs too large to fit in a single partition).

Fast method, since programs reside wholly in memory (no disk swapping) and can make optimal use of system resources.

Both CLASS I/O and FORTRAN COMMON may be used for communication between programs.

the obvious choice is to take advantage of the HP21MX operating system's ability to perform multitasking, allowing the system to share its resources among a number of concurrently executing programs. Those programs may then communicate either through a special type of memory common known as system common (Ref 39:1.6)), or through calls to an operating system message-passing utility (a service known as CLASS I/O (Ref 39:1.10)).

3.3.4 -- Interprocess Communication Alternatives

While it is clear that CLASS I/O and system common are the best means of communication between concurrently executing application programs, it is not clear how data should be held and passed by the various user interface modules (which together form a single program).

A memory space problem arises because of the user interface's need to retain all of the default and current values for each of arbitrarily many menus (if current values were not retained, then the user would lose any changes made to the default values of a given menu if he left that menu page without calling for its execution). Default values are kept in the MD(NNN) files, and so need not be kept in memory. It would be preferable to keep the current values in memory, so that file access wouldn't be necessary each time the user changed menu pages. However, in light of the fact that a single menu page's entries could well amount to hundreds of characters, this would restrict future expansion potential

rather severely. Therefore, each menu page's current values are kept in a CV(NNN) file. Since the HP21MX system has a hard disk, access time should be acceptable.

With the current values stored in files, there is no need for the explicit passing of dozens of variables which would otherwise be required. Another benefit gained is reduced data name duplication. For instance, many of the menus require a "Number of data points" entry. If all current values were retained in memory, each would have to be assigned a distinct name so that consistency checking could be performed (the user may actually want to have one module providing more data points than another is accepting). This could lead to a lot of confusion, and ultimately to nearly meaningless data names. Keeping current values in files thus allows the same consistent data name to be used with the same data item throughout the system, with different values given their full meaning through context (the name of the file in which they're stored).

3.3.5 -- Physical Design Structure Charts

In this subsection, Structure Charts with accompanying remarks are presented for the entire system. Many of the Structure Charts (especially those at the higher levels) contain modules which have already been described at some length in the logical design section (Section 3.1); for those, all that is added here is module and variable names

that correspond to their actual program counterparts. Other modules are presented for the first time in this section.

Many of the module and data names in the following section may seem a bit cryptic. The shortness of module and variable names in all physical descriptions of the system is an unfortunate consequence of using HP's version of the FORTRAN programming language, which (as is the case with most FORTRAN's for variable names) restricts variable names to at most six characters and program/subroutine names to at most five unique characters. It may be helpful to refer to the Data Dictionaries (Appendices F and G) when trying to relate logical names (from DFD's) to the actual shortened physical names used in Structure Charts.

3.3.5.1 -- DSP Structure Chart

The first structure chart, DSP (Figure 12) provides the same global view of the system's physical structure as the context DFD did of the system's logical structure in subsection 3.1.1. All of the modules on this chart perform almost pure managerial functions. They instigate and coordinate the activities of the lower-level modules doing the actual work. The DSP module, at the top, calls first GREQ and then SAREQ again and again to (respectively) get and then satisfy user requests until the user asks to exit the system or a fatal error occurs. GREQ first calls GOPT to get the user's options (building blocks of a request), then EOPT to edit the requests, and finally (if the options passed

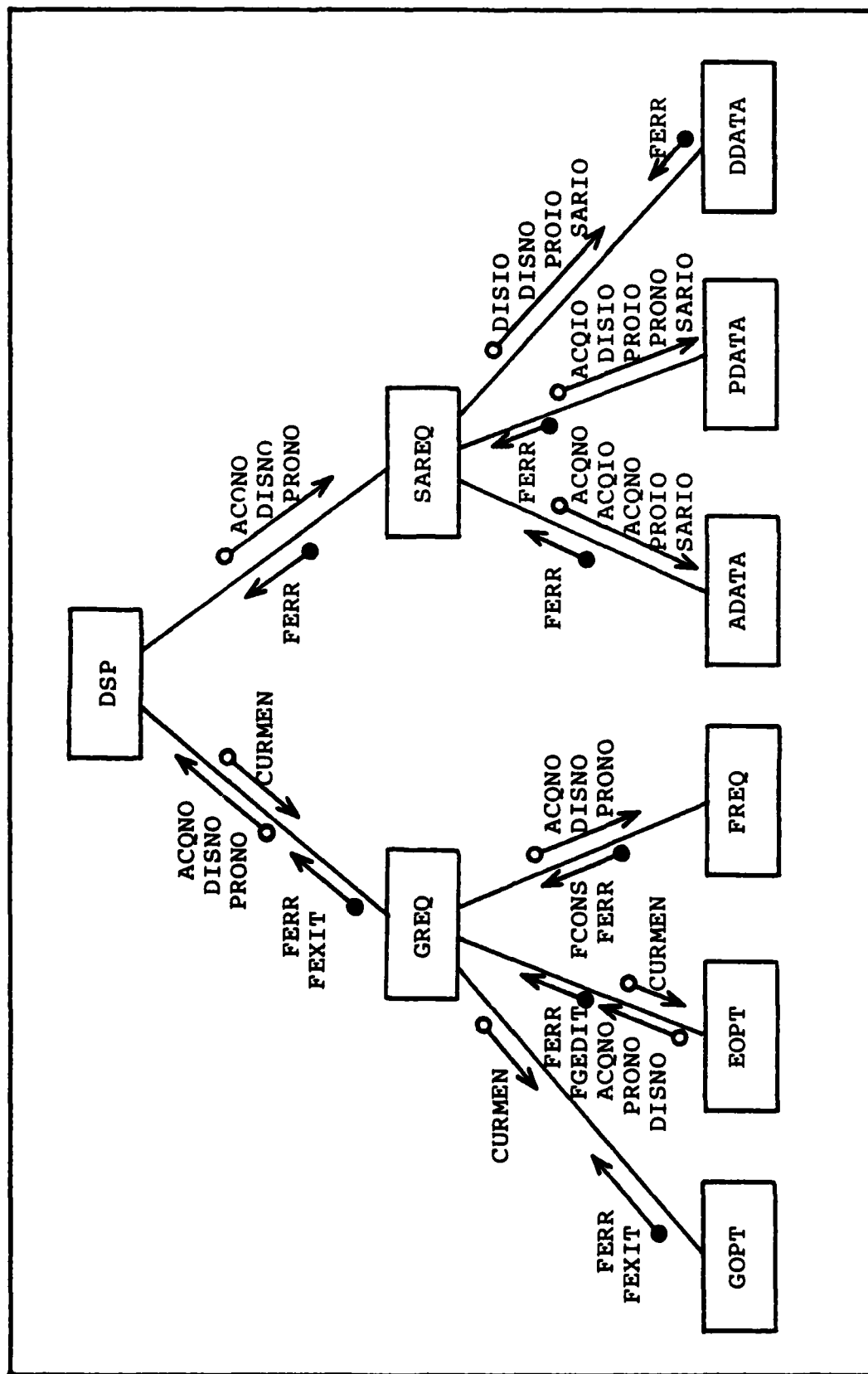


Figure 12
DSP Structure Chart

their edits) FREQ to format the disparate options into coherent requests which are then passed in files to application programs to direct their activities. Once GREQ has successfully gotten the user's request for services, DSP calls SAREQ to satisfy the request. SAREQ looks at the parameters it has been provided and determines the nature of the user's request. If data acquisition (e.g. A/D conversion) is to be performed, ADATA is called to start the appropriate data acquisition application program running. If data processing (e.g. data transform) is to occur, PDATA is called to start the appropriate data processing application program(s) running. And if data display (graphics, etc.) is part of the request, DDATA is called to start the proper data display application program running. All of the application programs are provided with information that allows them to communicate with SAREQ and with the other concurrently running programs, and are held in an idle state until each of the other application programs have been initiated and are ready to accept/provide data.

3.3.5.2 -- GOPT Structure Chart

The next structure chart, GOPT (Figure 13), shows the major modules involved in writing prompts to the terminal screen and accepting the user's responses (and default values) returned. Each time GOPT is called, it first calls DMENU to display a menu. Once the menu has been displayed, AOPT is called to poll the terminal and accept the block of

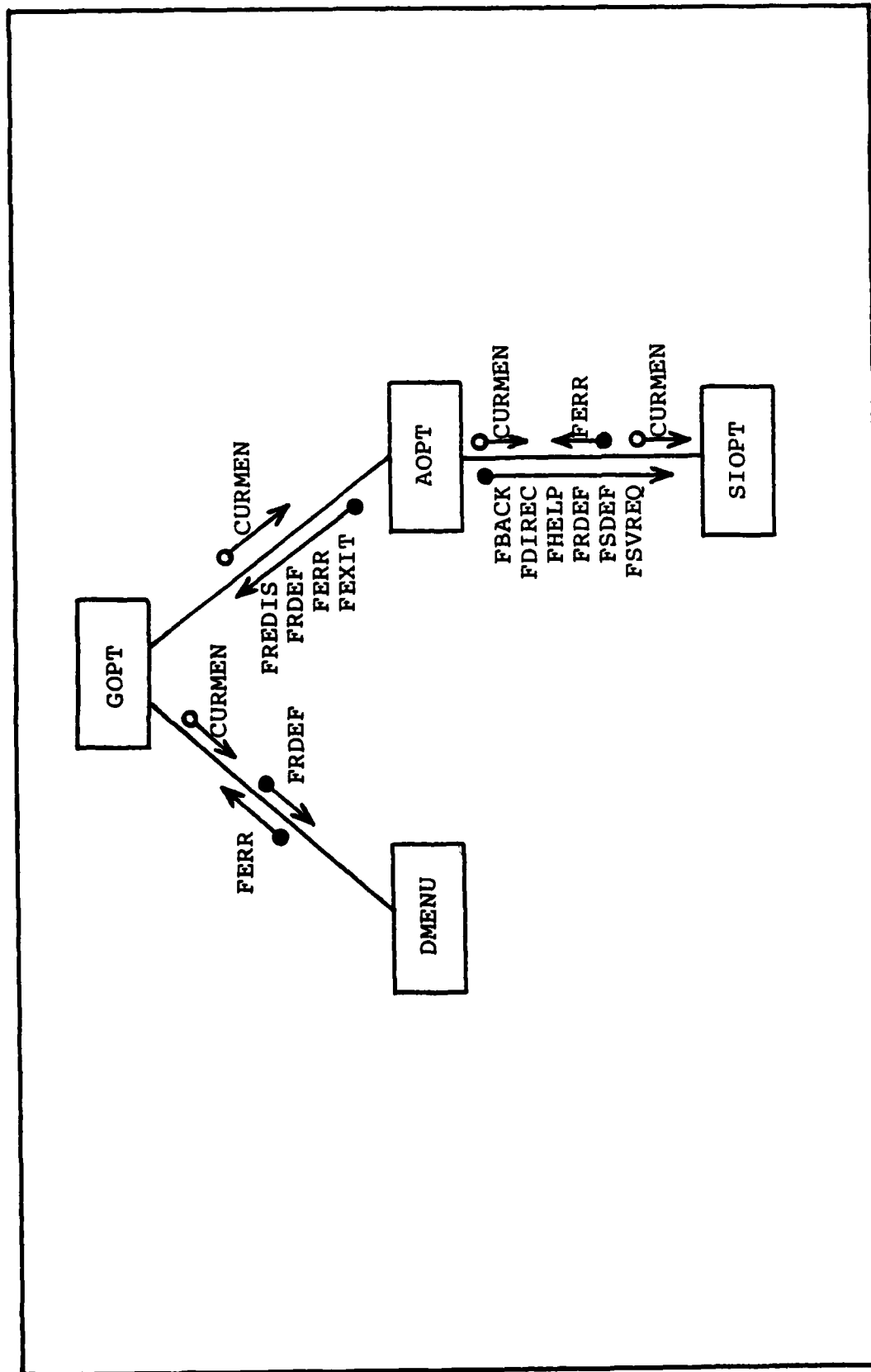


Figure 13
GOPT Structure Chart

options returned. Before returning control to GOPT, AOPT checks to see if the user initiated the data transfer by hitting one of the terminal's programmable function keys, which would indicate that the user wants to have some sort of an immediate service performed. If one of the function keys has been hit, and if it is one to which an immediate option meaning has been assigned (such as a request for help), SIOPT is called to satisfy the user's immediate request.

3.3.5.3 -- DMENU Structure Chart

The DMENU structure chart (Figure 14) shows the modules that take a menu definition stored in an MD(NNN) disk file and translate it into a screen menu. BORDR first draws a border around the screen. Then each of the PREC(N) modules is called in turn repeatedly to handle its respective MD(NNN) record type for each of arbitrarily many record groups. PREC1 reads type-1 records and positions the terminal's cursor according to row and column coordinate pairs found in them. PREC2 reads type-2 records and turns on the terminal's inverse video function if instructed to. PREC3 reads type-3 records and writes out the prompting text found on them. PREC4 reads type-4 records to find out the length of the field to be provided for the user's response to that particular prompt (which is also the length of the field to contain the system's default value for that prompt), and passes the length on to PREC5. PREC5 reads type-5 records to find out whether or not the user wants the default value to

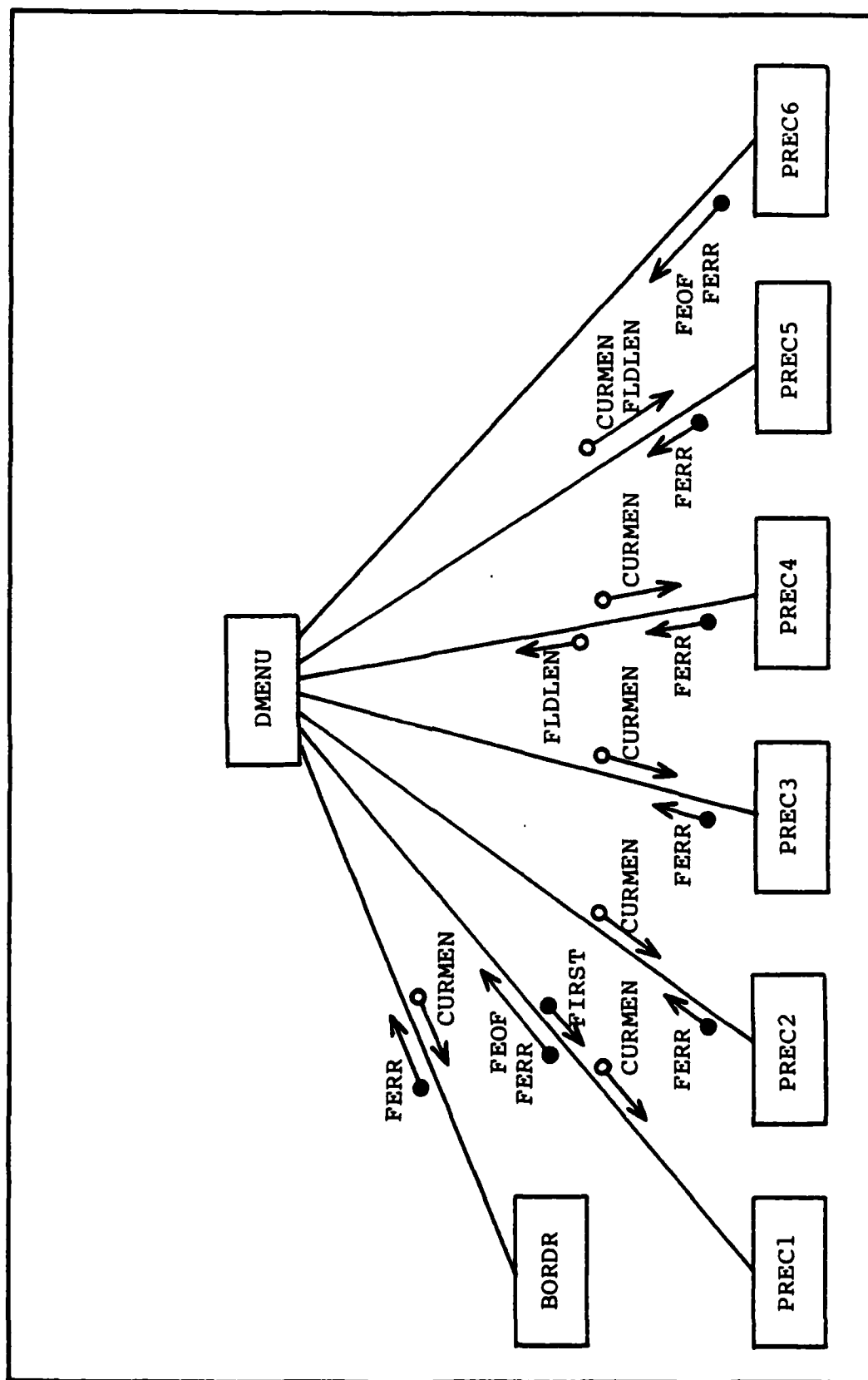


Figure 14
DMENU Structure Chart

be printed in inverse video, and sends attribute characters to the terminal accordingly along with the proper number of spaces for the default field. PREC6 is called repeatedly after all of the response fields have been set up, and reads type-6 records containing default values. PREC6 then writes the default values to the terminal, which automatically places them in the default fields in the order they are received.

3.3.5.4 -- SIOPT Structure Chart

The next structure chart, SIOPT (Figure 15), contains the modules responsible for satisfying immediate options such as requests for help or directory listings. SIOPT calls the appropriate module based on the values of flags passed to it by AOPT (see Figure 13). HELP is called to give additional information about the menu the user is faced with. The user may browse through the help text (there may be many pages) until ready to return to the menu that was left. DIREC provides listings of different file types to help the user in choosing or remembering a file name for some menu response. The directories are either sent to the terminal or placed on a file from which they may later be listed. SVREQ gathers all of the options that the user has entered so far (from the CV(NNN) files which have been receiving them) and forms them into a request file which may later be read and accepted as though the user had just typed in all of those same options. REREQ reads a request file created by SVREQ and places the

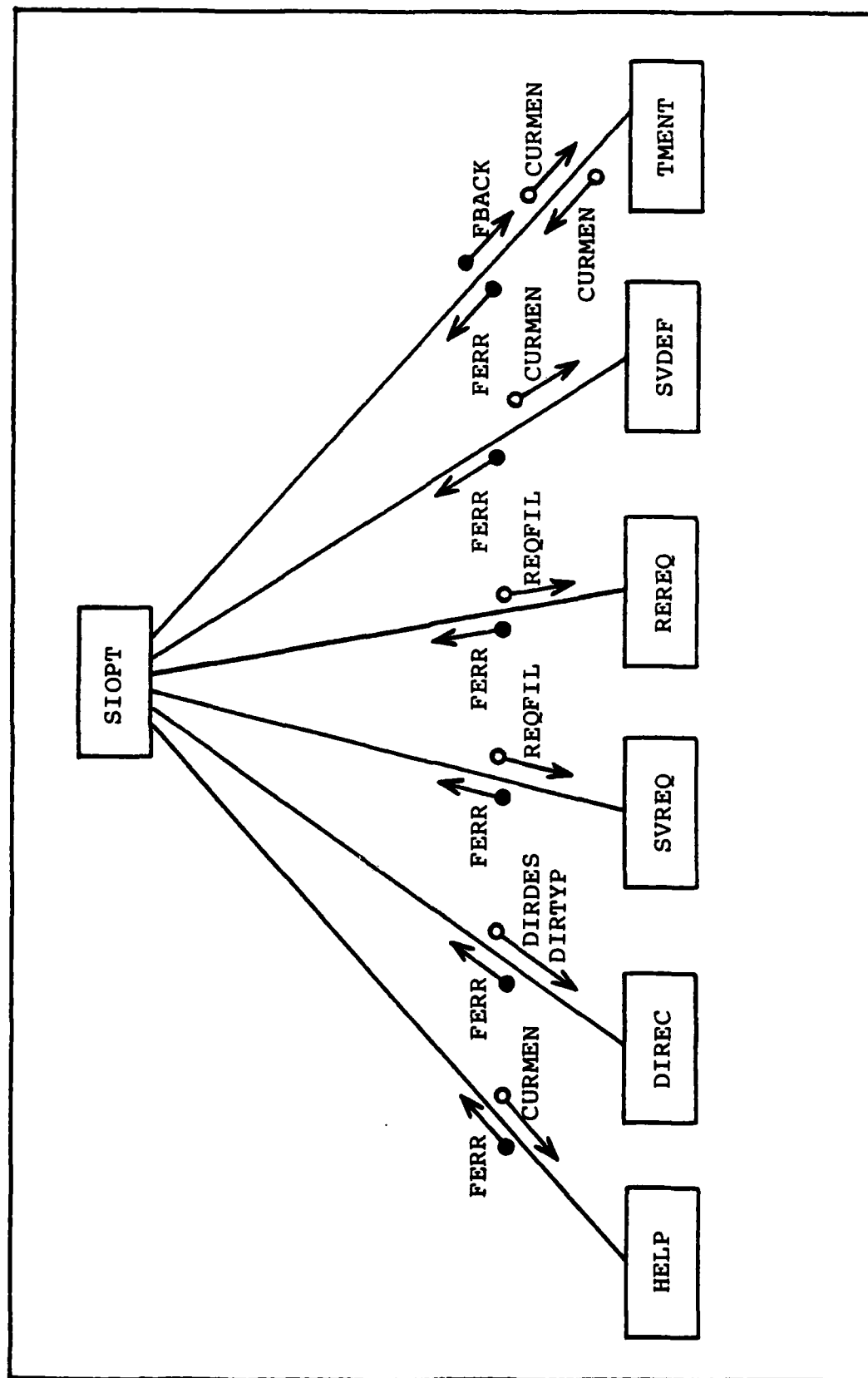


Figure 15
SIOPT Structure Chart

options it contains into appropriate CV(NNN) files so that by all appearances the user has just entered those options from the keyboard. SVDEF saves the system's current values (which are simply the default values originally read in with any changes made by the user) as its new default values. Finally, TMENT figures out which menu should be displayed next (each menu is assigned a unique number), according to whether more options are required to complete a pending request, whether the user has asked to back up one menu, etc. TMENT does this by examining a data file (MENUTR) which contains data associating the menus in a tree-like fashion.

3.3.5.5 -- EOPT Structure Chart

In the EOPT structure chart (Figure 16), we see that EOPT has a set of editing modules that it may call upon to check the validity of the user's responses in a particular menu. Each of the edit modules is uniquely associated with a single display menu. Thus EM001 is called to edit menu number 1, EM002 is called for menu number 2, etc. The EM(NNN) modules' only responsibility is to return an error flag (FGEDIT) signifying whether the options edited were good or not.

3.3.5.6 -- FREQ Structure Chart

The FREQ structure chart (Figure 17) shows the three groups of modules that FREQ controls, each of the groups being responsible for formatting the request for a particular type of application program, either acquisition, processing,

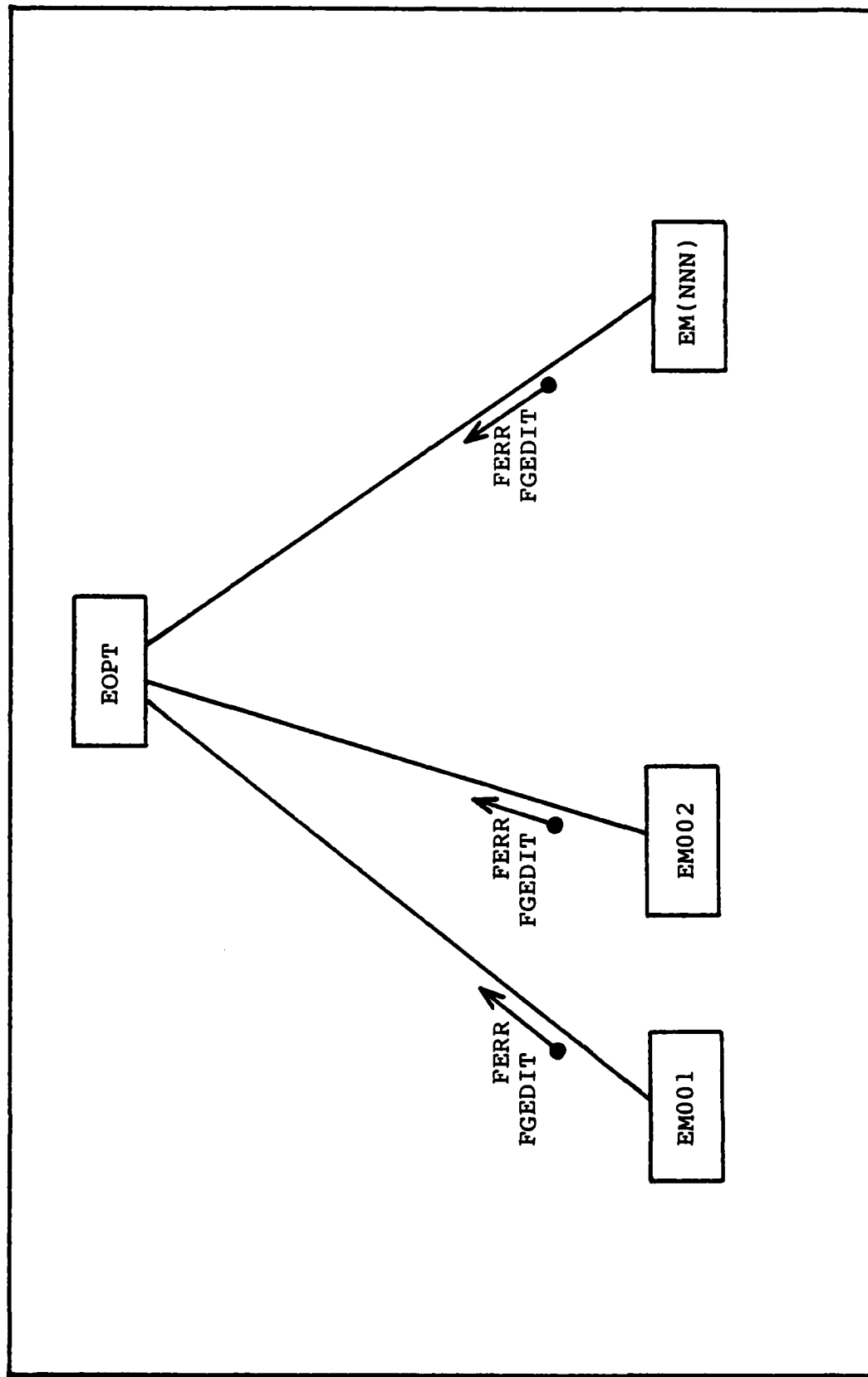
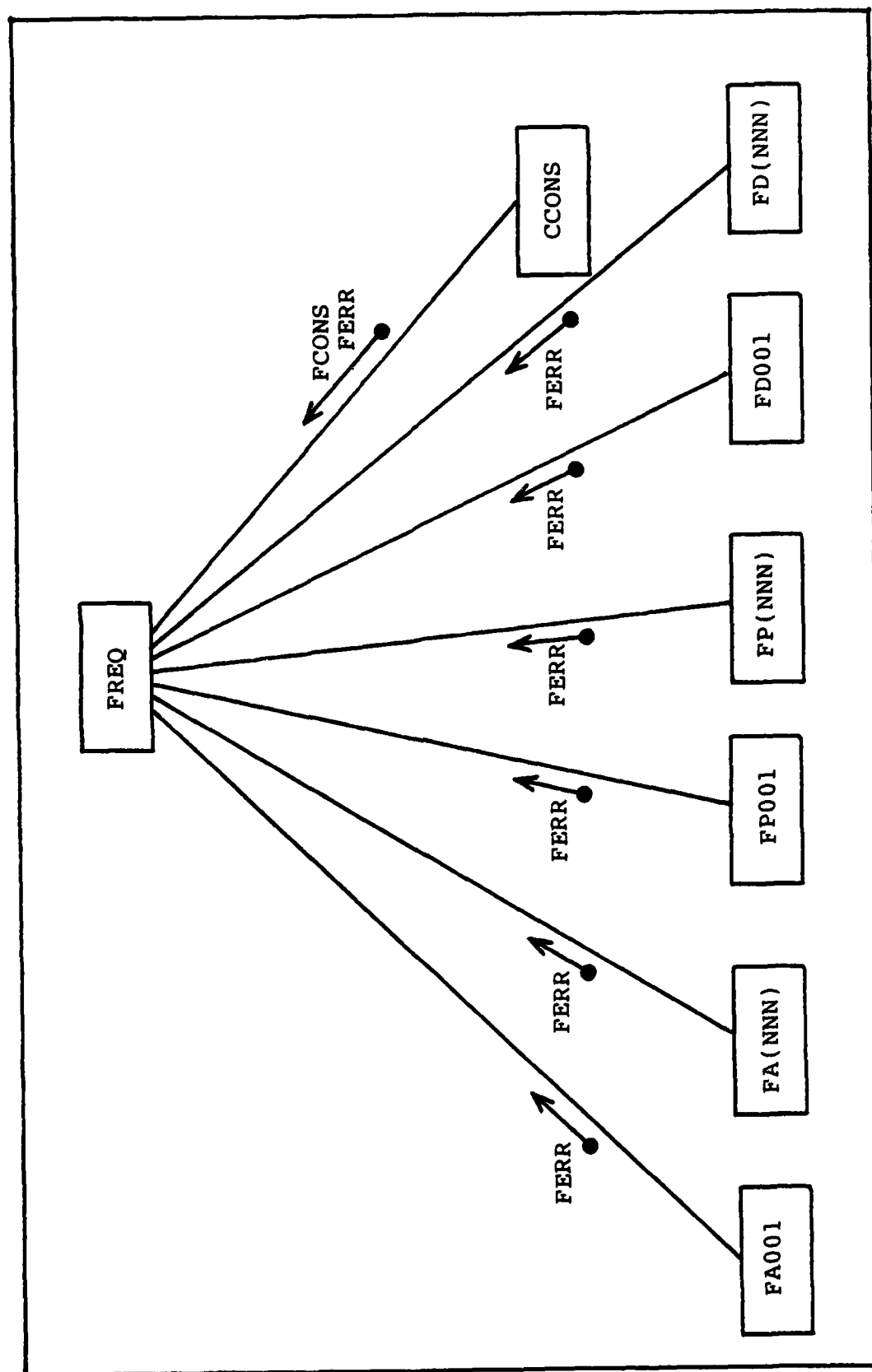


Figure 16
EOPT Structure Chart



or display. FREQ determines which of the modules it must call by examining the data items ACQNO, PRONO, and DISNO; they contain numbers identifying, respectively, the acquisition, processing, and display application programs which must be invoked to satisfy the request. The numeric portion of each formatting module's name uniquely associates the module with a particular application program (which has the same number as part of its name). The FA(NNN) modules format requests for data acquisition application programs. For example, FA001 formats data for the program AD001. The FP(NNN) modules format requests for data processing application programs. For example, FP005 formats requests for the data processing application program PD005. The FD(NNN) modules format requests for data display application programs. FD001 formats requests for the application program DD001, etc. The final module on the chart, CCONS, ensures that all of the various parts of the user's request are consistent. For example, if an acquisition application program is called upon to provide data to a processing application program, that processing program must have been specified as part of the overall user request. If it wasn't, and the user tries to execute the request, CCONS will point out the inconsistency and force the user to go back and reconcile it.

3.3.5.7 -- ADATA Structure Chart

ADATA is responsible for executing data acquisition application programs, as shown by the ADATA structure chart (Figure 18). ADATA determines which acquisition program is to be executed based on the value of the data item ACQNO. If ACQNO equals two, for instance, then the user has requested some function provided by acquisition program number two. ADATA invokes the requested program, passing it communication parameters (ACQIO and PROIO) that enable it to communicate with SAREQ (which remains in charge of the overall process) and any data processing acquisition program to which it is to provide its data.

3.3.5.8 -- PDATA Structure Chart

The PDATA structure chart (Figure 19) diagrams PDATA's control of the data processing application programs. PDATA looks at the data item PRONO (which is an array) and invokes each of the processing programs whose identifying number is found in PRONO. An arbitrary number of processing programs may be invoked during a given request satisfaction; each is provided with communication parameters so that it may pass data and control information back and forth with other processing application programs and with SAREQ (things such as processed data and completion signals).

3.3.5.9 -- DDATA Structure Chart

The last structure chart, DDATA (Figure 20), shows the control relationship between DDATA and all of the data

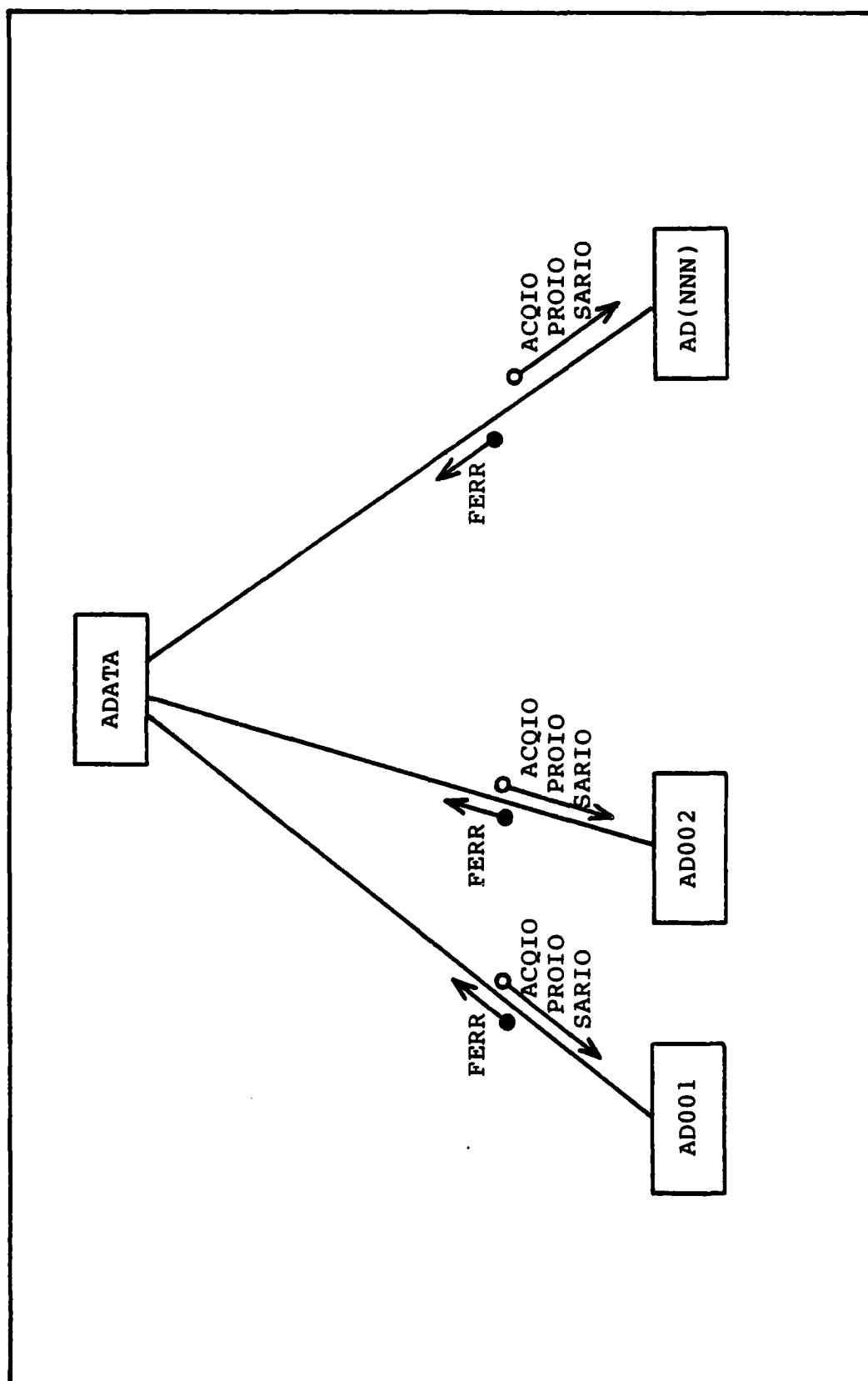


Figure 18
ADATA Structure Chart

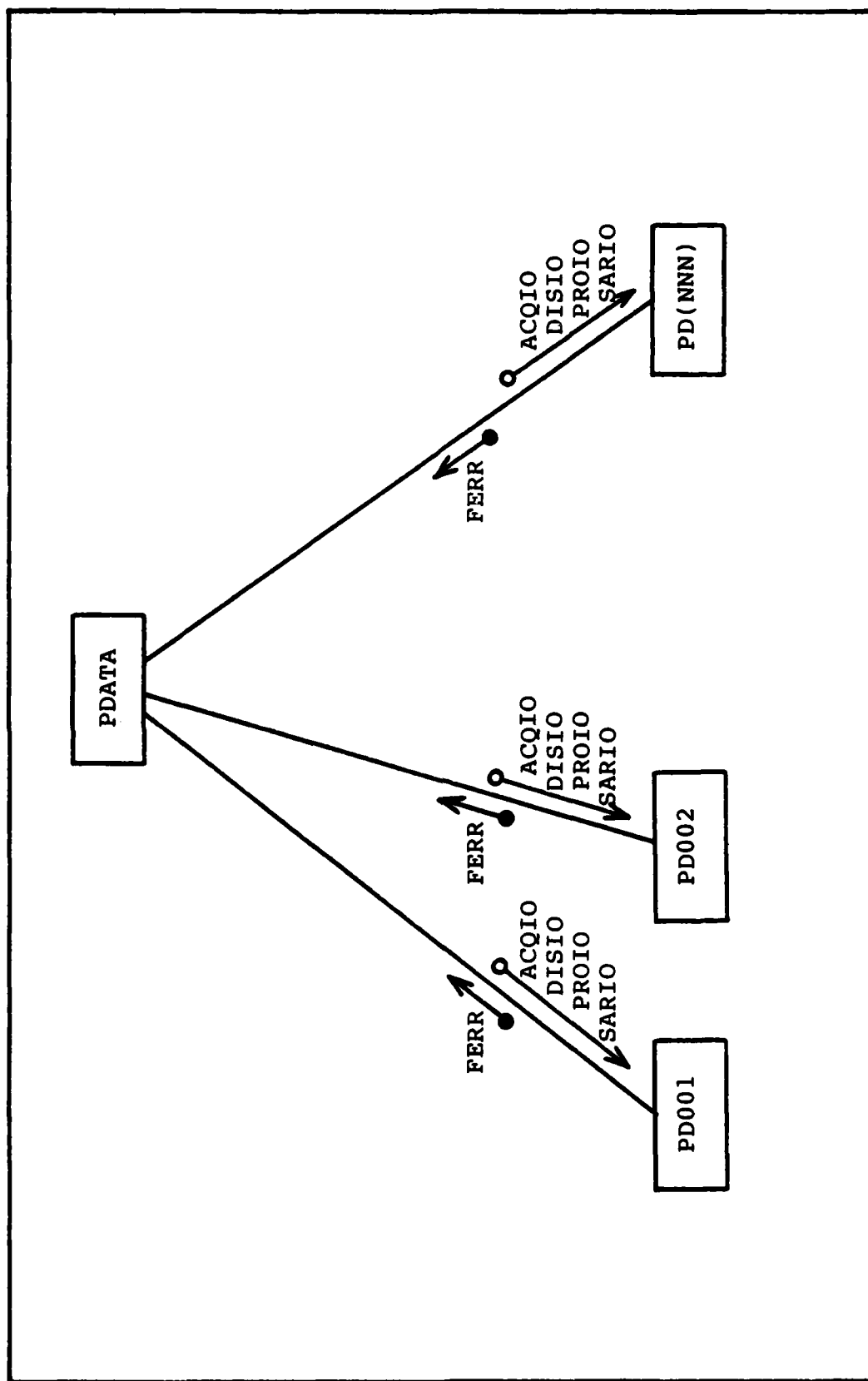


Figure 19
PDATA Structure Chart

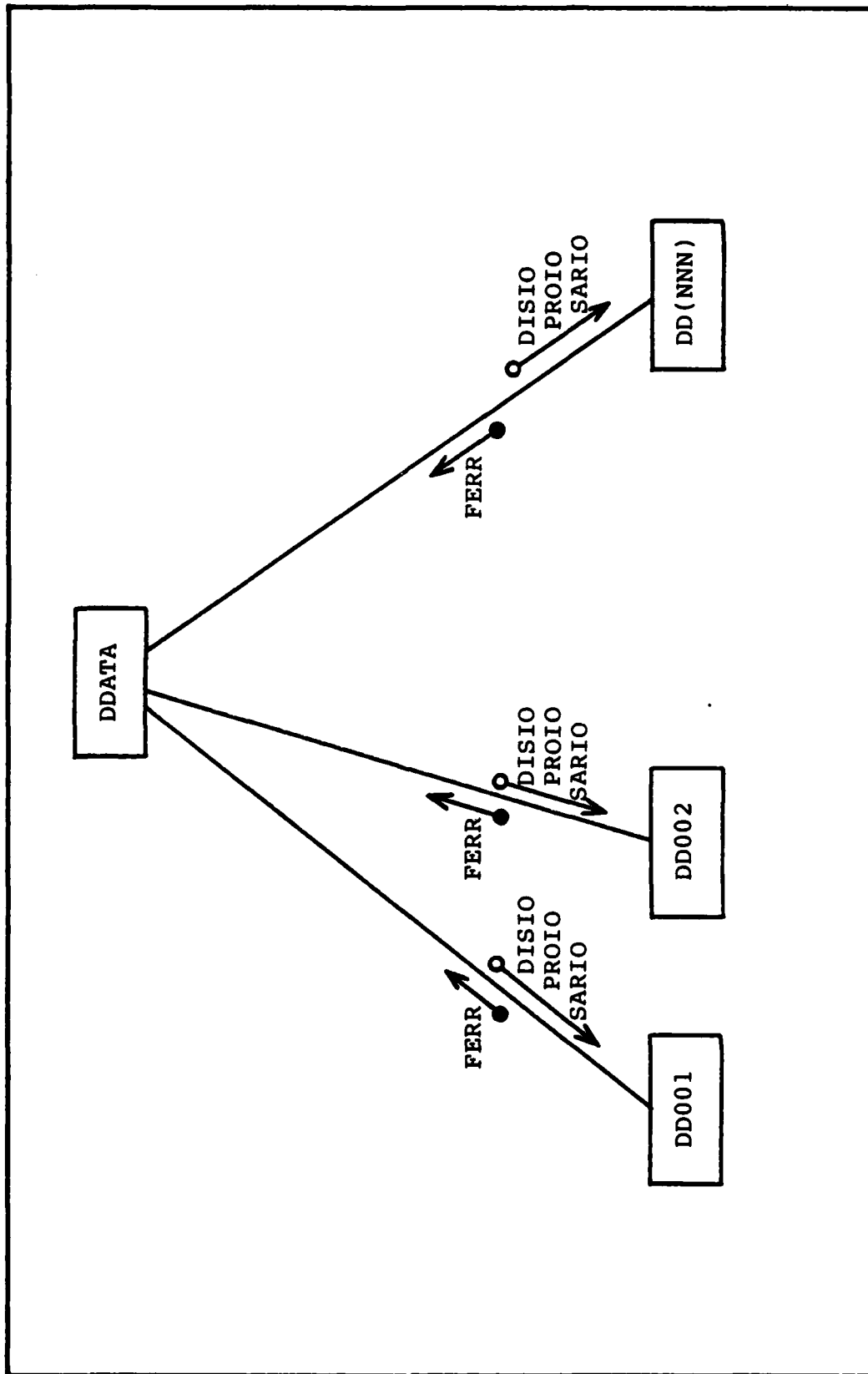


Figure 20
DDATA Structure Chart

display application programs. As with ADATA and PDATA, DDATA is responsible for executing application programs requested by the user. Which one is executed depends upon the value of the data item DISNO, which should be equal to the numeric portion of one of the display application programs' names. Again, communication parameters are passed to the display program that is invoked so that it may be controlled by SAREQ and (in this case, possibly) accept data from data acquisition or data processing programs.

3.4 -- Hardware Components

This section describes the makeup and configuration of the major hardware components needed to support the DSP software system at least minimally. All of the hardware described here was present at the time system implementation began. Recommendations for additional hardware are presented in Chapter 5, and Appendix C contains an exhaustive list of all the system's hardware.

The following paragraphs list the system's major hardware components along with brief explanations of their importance.

Computer: The system's central processor is an HP 2108A 16-bit microprogrammable minicomputer with 80K words of memory. Its Direct Memory Access (DMA) capabilities and Input/Output (I/O) flexibility are essential for real-time applications. DMA permits data to be acquired and placed in memory by the computer's Dual Channel Port Controller (DCPC), freeing the Central Processing Unit (CPU) to perform concurrent, independent tasks. The computer's I/O facilities (in particular its rearrangeable I/O card slots) allow a wide range of external devices to be interfaced directly to the system's main data bus, which means that DMA data transfers and low-level device interaction with the system are easily implemented (both are common requisites for real-time systems) (Ref 23).

Mass-Storage: The system has an HP 7906 hard disk drive. This is a 20-megabyte unit with two platters, one

fixed and one removable. Using DMA, this unit provides and accepts data at rates that make real-time data acquisition and storage a possibility. Its speed also makes feasible the use of data files for relatively fast communication between executing processes -- a crucial ability in a system handling large amounts of data whose limited address space (17K words at present) makes concurrent processes almost unavoidable.

Console Device: An HP2648A graphics terminal serves as the system's console device. This is the device through which the user interacts with the DSP system's user interface. Many of the system's objectives could be realized with a less capable terminal, but the abilities of performing direct cursor addressing, defining protected fields, and performing block data transfers are necessary for the menu-driven interface format that best meets the system's ease-of-use goal.

Hardcopy Output: A Heathkit H-25 printer allows the user to produce high-quality hardcopy output at a reasonable speed (60 characters per second). Without such a printer, documentation and system development efforts would be much more difficult, much less reliable, and much more time consuming.

3.5 -- Data Acquisition Application Program Characteristics

This section describes the characteristics of application programs (as opposed to user interface programs) responsible for acquiring the DSP system's data. Data is

acquired either from external sources, such as an Analog-to-Digital (A/D) converter, or from an internal generating function, such as a random number generator or FORTRAN library sinusoid. Each of the programs is capable of passing its data to a concurrently-executing program (for processing or display) or to a data file.

3.5.1 -- AD001 (Generate Data)

The data acquisition program known as AD001 generates data using a two-term linear combination of sinusoids. The user's options include: the number of data points; the scale of each sinusoid term; sinusoid frequency; sampling frequency; and phase shift.

The SIN data generated is defined by

$$(1) \quad f(n) = [a\text{SIN}(2\pi n/b + \theta_1) + c\text{SIN}(2\pi n/d + \theta_2)] ,$$

where N is the number of data points, a and c are scaling factors, and θ_1 and θ_2 are phase shifts. b and d are related to the sinusoid frequencies w_1 and w_2 and to the sampling frequencies F_{S1} and F_{S2} (all input by the user as options) by

$$(2) \quad w_1 = F_{S1}/b \quad \text{and}$$

$$(3) \quad w_2 = F_{S2}/d .$$

3.5.2 -- AD002 (Sample A/D Converter)

AD002 acquires data for the DSP system by sampling the analog signal via an A/D converter. The user's options are:

sampling rate (Hz); number of sampling points or continuous sampling; and data to file or directly to concurrent application program.

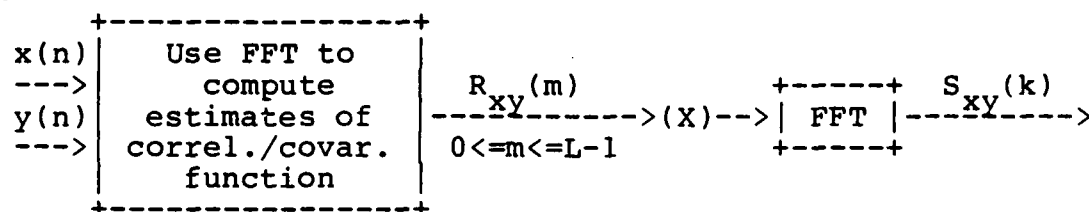
3.6 -- Data Processing Application Program Characteristics

This section describes the characteristics of certain programs taken from the IEEE's body of published DSP software (Ref 13). These programs process the data acquired by the data acquisition application programs of Section 3.4, and so are known as data processing application programs (as opposed to data acquisition or display application programs). They are termed application programs to distinguish them from the system's user interface. Any changes made to these programs during implementation are noted in Chapter 5. Of course, all of the programs will have to have at least I/O, communication, and coordination code added. The programs are treated here in essentially the same order as they occur in Table III. They are identified by both their Reference 13 and DSP system names.

3.6.1 -- PD001 (Correlation and Covariance)

The first application program is from (Ref 13:2.2), and has the name PD001 in this system (it is called CMPSE in Reference 13). It provides an estimate of the auto-correlation function of a signal, the auto-covariance of a signal, the cross-correlation between two signals, or the cross-covariance between two signals, using FFT techniques. All of these functions are time-domain measures of the amount

of similarity between a signal and itself ("auto-") or between it and another signal ("cross-"). An estimate of the power spectrum of the signal (obtained by computing the discrete Fourier transform of a windowed version of the correlation function) is also provided. Figure 21 shows a block diagram of the signal processing within the program. "(X)" is the windowing operator $w(m)$.



(Figure 21)
Block Diagram of Correlation/Covariance Method

In the general case there are two input signals, $x(n)$ and $y(n)$, which are defined for the interval $0 \leq n \leq N-1$ (N being the total number of data points provided). Outside this interval $x(n)$ and $y(n)$ are assumed to be zero. The correlation function R_{xy} to be computed is defined as

$$(4) \quad R_{xy}(m) = (1/N) \sum_{n=0}^{N-1-m} (x(n) - \bar{x})(y(n+m) - \bar{y}) ,$$

where $0 \leq m \leq L-1$, L is the desired number of correlation values (known as lags), and \bar{x} and \bar{y} are the estimated means of $x(n)$ and $y(n)$, i.e.,

$$(5) \quad \bar{x} = (1/N) \sum_{n=0}^{N-1} x(n) , \text{ and}$$

$$(6) \quad \bar{y} = (1/N) \sum_{n=0}^{N-1} y(n) .$$

For covariance function estimates, these means are subtracted in Equation (4). For correlation function estimates, the means are not subtracted in Equation (4). To obtain the power spectrum estimate from the correlation function, one of two procedures is used. For $x(n) = y(n)$, i.e., auto-correlation/covariance, the windowed auto-correlation function $R_{xx}^*(m)$ is created as:

$$(7) \quad \begin{aligned} R_{xx}^*(m) &= R_{xx}(m)w(m) & 0 \leq m \leq L-1 \\ &= R_{xx}(M-m)w(M-m) & M-L+1 \leq m \leq M-1 \\ &= 0 & \text{otherwise} \end{aligned}$$

where $w(m)$ is a symmetric finite duration window which tapers $R_{xx}(m)$ to 0 as M (the size of the FFT) tends to $L-1$. If $x(n) \neq y(n)$, i.e., for cross-correlation or covariance, the signal $R_{xy}^*(m)$ is created as:

$$(8) \quad \begin{aligned} R_{xy}^*(m) &= R_{xy}(m)w(m) & 0 \leq m \leq L-1 \\ &= 0 & \text{otherwise} \end{aligned}$$

where $w(m)$ again represents a finite duration window (either rectangular or Hamming).

3.6.2 -- PD002 (Coherence)

The second application program is known as CCSE in (Ref 13:2.3), and as PD002 in the DSP system. It provides estimates of auto and cross power spectral density functions together with coherence and generalized cross correlation functions. For the purposes of this system, it is useful because it is the only program in Reference 13 to provide coherence function estimates. Other smaller, less complex programs are used to provide the other functions it is capable of. PD002's first step is to segment the two digital waveforms it is provided into N segments, each having P data points. Second, each segment is then multiplied by a smooth weighting function. Third, the z-transform of the weighted P-point sequence is evaluated on the unit circle in the Z-plane. The two-sided z-transform of an infinite sequence may be defined as

$$(9) \quad X_n(z) = \sum_{p=-\infty}^{\infty} x_n(p) z^{-p} \quad , \quad n = 1, 2, \dots, N$$

where z is any complex variable. $Y_n(z)$ is defined similarly. Since $x_n(p)$ and $y_n(p)$ are typically finite in duration, the infinite series in Equation (9) becomes finite. Evaluation of the z-transform at P equally spaced points around the circle yields the discrete Fourier transform or DFT

$$(10) \quad X_n(k) = \sum_{p=0}^{P-1} x_n(p) e^{-j2\pi pk/P} .$$

In the fourth and final stage, the discrete Fourier coefficients $X_n(k)$ obtained in the third step are used to estimate the elements of the power spectral density matrix by averaging "raw" power spectral estimates over all the N segments. The magnitude squared coherence estimate we are interested in is then computed from the spectral estimates. The spectral estimates are defined as

$$(11) \quad G_{xx}(k) = c_g \sum_{n=1}^N |X_n(k)|^2 ,$$

$$(12) \quad G_{yy}(k) = c_g \sum_{n=1}^N |Y_n(k)|^2 , \text{ and}$$

$$(13) \quad G_{xy}(k) = c_g \sum_{n=1}^N X_n^*(k) Y_n(k) ,$$

with the constant gain

$$(14) \quad c_g = \frac{1}{(N f_s P)} ,$$

where f_s = sampling frequency and "*" indicates complex conjugation. The estimate of magnitude squared coherence (MSC) is

$$(15) \quad C_{xy}(k) = \frac{|G_{xy}(k)|^2}{[G_{xx}(k) G_{yy}(k)]} .$$

3.6.3 -- PD003 (Convolution)

The third data processing application program in this system, PD003, is an FFT-based convolution program taken from (Ref 13:3.1) where it is known as FASTFILT. PD003 uses the overlap-add method to perform the digital filtering operation. PD003 filters the digital signal provided to it according to a set of impulse response values that the user defines. In general, the convolution operation may be defined on two time-domain signals $x(n)$ and $h(n)$ as

$$(16) \quad x(n)*h(n) = \sum_{k=0}^{N-1} x(k)h(n-k) ,$$

where "*" is the convolution operator and N is the number of signal values. Three basic steps are involved in convolving a pair of signals. First, the Discrete Fourier Transforms (DFT's) of the two signals are computed using a Fast Fourier Transform (FFT) algorithm. Second, the transforms of the signals are multiplied together at all pertinent frequency points. Third, the inverse transform of the product is computed, again using an FFT algorithm, producing the desired convolved output signal (Ref 47:285). Since this program uses the convolution method to perform filtering operations on a single time-domain signal rather than to perform general convolutions on pairs of time-domain signals, it expects not two time-domain signals but rather a time-domain signal $x(n)$ and the frequency-domain description of the impulse response characteristics of the filter to be applied to the signal.

Thus $h(n)$ in Equation (16) is supplied to the program having been transformed already to the frequency domain by something such as a DFT, or having been specified directly by the user.

3.6.4 -- PD004 (FFT)

The fourth data processing application program, PD004, is an implementation of the Fast Fourier Transform (FFT) algorithm. The FFT is, as its name suggests, a fast method of performing the Discrete Fourier Transform (DFT). The key to its speed lies in the identification and systematic elimination of redundancy in the calculations executed by the DFT (Ref 10:408). A large number of different FFT programs are available in Reference 13. Each favors a particular set of functional capability decisions, with most of the decisions requiring that a tradeoff be made. For example, the most efficient FFT programs (in terms of execution speed) are also the ones that require the most memory space. Likewise, being able to operate on complex-valued input signals or on signal sets not having some power-of-two number of elements adds to the complexity and size of the programs. PD004 is one of the smaller FFT programs, known as FAST (Ref 13:1.2-1). It operates on real-valued input data, and the number of data points provided must be a power of two (such as $2^{10} = 1024$). The operation performed by PD004 may be defined as

$$(17) \quad X(k) = \sum_{n=0}^{N-1} x(n)e^{-i2\pi nk/N} ,$$

where $x(n)$ is real, $X(k)$ is complex, and $k=0,1,\dots,N/2$. Only $(N/2+1)$ complex DFT values need to be computed and stored because $x(n)$ is real and by symmetry

$$(18) \quad X(N-k) = X^*(k) ,$$

for $k = 0,1,\dots,N/2$ with "*" denoting complex conjugation.

3.6.5 -- PD005 (FIR Filter Design)

The fifth data processing application program, PD005, is a program for designing Finite Impulse Response (FIR) digital filters. Its Reference 13 name is EQFIR (Ref 13:5.1). PD005 uses the Remez exchange algorithm to design linear-phase FIR digital filters with minimum weighted Chebyshev error in approximating a desired ideal frequency response. It offers some of the more common ideal filter types such as multi-band, bandpass, Hilbert transform, and differentiators. The frequency response of a general FIR digital filter with an N -point impulse response $\{h(n), n = 0,1,\dots,N-1\}$ is the z -transform of the sequence evaluated on the unit circle, i.e.,

$$(19) \quad H(f) = H(z) \Big|_{z = e^{j2\pi f}} = \sum_{n=0}^{N-1} h(n) e^{-j2\pi n f} .$$

The frequency response of a linear-phase filter (the type PD005 designs) may be written as

$$(20) \quad H(f) = G(f) \exp \left\{ j \left[\frac{L}{2} \pi - \left(\frac{N-1}{2} \right) 2\pi f \right] \right\} ,$$

where $G(f)$ is a real-valued function and $L = 0$ or 1 . There are four cases of linear-phase FIR filters to consider, differing in the length of the impulse response (even or odd) and the symmetry of the impulse response. The form of $G(f)$ in Equation (20) depends upon which of the four cases is being used. All four of the cases are combined into a single algorithm in this program, so that a only one central computation routine (based on the Remez exchange method) is needed to calculate the best approximation in each of the our cases.

3.6.6 -- PD006 (IFFT)

The sixth application program (PD006) is an implementation of the Inverse Fast Fourier Transform (IFFT) algorithm. The IFFT is a fast means of computing the Inverse Discrete Fourier Transform (IDFT). It relies on the identification and elimination of redundant computations for its speed (just as the FFT does). PD006 is taken from Reference 13, where it is known as FSST (Ref 13:1.2-1). As in the case of the FFT, there are a number of IFFT programs to choose from. PD006 is similar in its capabilities and level of complexity to PD004 (the FFT program). It expects complex-valued input data, provides real-valued output data, and the number of signal data points to be transformed must be some power of two. The IDFT may be defined as

$$(21) \quad x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N} ,$$

where $X(k)$ is complex, $x(n)$ is real, and $n = 0, 1, \dots, N-1$. Since $X(k)$ has complex conjugate symmetry around $k = N/2$, only $(N/2+1)$ complex values of $X(k)$ are actually needed to compute the N values of $x(n)$.

3.6.7 -- PD007 (IIR Filter Design)

PD007, the seventh data processing application program, is a design program for Infinite Impulse Response (IIR) filters. It is found in Reference 13 under the name of EQIIR (Ref 13:6.1). PD007 is capable of handling these filter types: lowpass, highpass, symmetrical bandpass, and bandstop. The types of approximation available are the maximally-flat magnitude (Butterworth), the Chebyshev types I and II, and the elliptic (Cauer) forms. In addition to providing filter approximations, the program also minimizes coefficient wordlength, optimizes noise performance, and allows the user to analyze filter performance by providing a cascade-realization transfer function.

3.6.8 -- PD008 (Waveform Averaging)

PD008 performs waveform averaging. It accepts a number of sets of data values (each of these sets being called a frame) and maintains a point-by-point running average. The frames may be from a data file, or they may be provided as real-time data by a data processing application program. PD008 is useful for allowing comparisons between a signal's long-term average characteristics and its single-frame characteristics. Since PD008 can supply its output to either

a data file or another application program, a graphical picture of a signal's gradually stabilizing average may be formed in real time by sending PD008's output to one of the data display application programs (DD(NNN)).

PD008 is the only data processing application program described in this section that is not taken from Reference 13.

3.7 -- Data Display Application Program Characteristics

This section describes the application programs responsible for displaying DSP data to the available HP21MX DSP system graphics devices. The programs can take their input from either a concurrently-executing application program (acquisition or processing) or from data files. Each of the programs is tailored to work with a particular graphics device.

3.7.1 -- DD001 (Display to HP2648A)

DD001 takes DSP data and displays it graphically on the HP2648A terminal. The user's options include: number of data points per block; number of blocks; real or complex data; and if complex data, display real, imaginary, or magnitude values.

3.7.2 -- DD002 (Display to HP1310)

DD002 takes DSP data and displays it graphically on the HP1310 terminal. The user's options include: data from file or from a concurrently executing application program; X and Y axis labels; number of grid lines per axis; and minimum and maximum X and Y values (again required only if scaling is desired).

3.8 -- General Testing Procedures and Criteria

This section describes the general procedures to be followed and the universal criteria to be met by all software modules during system testing. A complete test plan,

including more specific tests directed toward individual modules, together with the results of all tests, is found in Appendix J. The testing philosophy adopted in this thesis effort is that "Testing is the process of executing a program with the intent of finding errors" (Ref 32:5).

Testing will be performed on modules as they are implemented (in accordance with the top-down method of system development), starting with the top-most control modules in the system's structure (see the Structure Charts in Subsection 3.3.5) and proceeding level by level. All of the procedures and criteria in this section are to be applied to every module, without regard for the detailed logic the module may contain. Such testing is known as "black box" testing (Ref 32:8-9), in light of the fact that the module is treated as a box closed to the view of the tester. The more extensive tests given in Appendix J are designed with a full knowledge of each module's contents, and are thus known as "white box" tests (Ref 32:9-11). In general, the black box tests concentrate on catching commonly occurring errors, such as the failure to properly handle input values located on the boundary of a range of permissible values, or the inability to properly handle termination/error conditions. White box tests, on the other hand, probe more than just the obvious gross functionality of a module. They are meant to locate subtle, unpredictable errors that are usually the result of combinations of coincidental oversight in the detailed logic structure of the module. Such errors are often discovered by

exercising the module with data that causes the module's code to be executed exhaustively -- every possible path (and perhaps combination of paths) in the code is executed. Unfortunately, truly exhaustive testing is virtually impossible with non-trivial systems (Ref 32:8-11). For the purposes of this system, an intelligent balance between completeness and time required will be sought by choosing those test methods which have been shown to have the highest probability of discovering errors (Ref 32:36-37), while coming as close as possible to exhaustive testing.

Once a module is coded and compiles without errors, the procedure for testing it is as follows:

- 1) The module is examined to ensure that it is completely, clearly, and accurately documented. This is done with the belief that internally documenting a system is extremely important (especially with a system that's evolving), and is much more difficult, too easily slighted, and error-prone if put off until the system is complete.

- 2) All modules subordinate to the module being tested are "stubbed". That is, dummy subordinates are created to accept and return the parameters needed to permit the proper operation and evaluation of the module undergoing testing.

- 3) A set of test cases is defined. Each test case should specify the conditions under which the test is to take place, the data to be provided the module (an exact image if that is practical), and the exact behavior (especially in the case of error conditions) and results expected from the

module. The characteristics of any required stubs should be included. The tests should cover both white and black box cases.

4) The tests are performed, with changes made to the module until it performs satisfactorily. Each change or group of changes made to a module results in a new version number being assigned to the module. The module's final version number and the date on which the tests are successfully passed are then noted in Appendix J.

5) The module is installed into the system, and testing continues with any modules that remain at that level in the structure chart before proceeding to the next level. Any subsequent changes (other than purely cosmetic) made to a validated module will require it to be retested.

These are the black box tests every module should be able to successfully pass (as applicable):

1) Every data item provided to the program should be assigned values at the lowest, mid, and highest limits of its range, if it is to be so restricted. This is known as boundary-value analysis (Ref 32:50-55).

2) For each input data item which may occur arbitrarily many times, a test should be performed with no occurrences, a typical number of occurrences, and a large number of occurrences of that data item (this is also a part of boundary-value analysis).

3) All numeric items should be assigned alphabetic values, and all alphabetic items should be assigned numeric

values to test the error-handling abilities of the module.

4) For each file the module uses, a test should be performed with the file in existence and without the file in existence to test the error-handling abilities of the module.

5) For each external device the module uses, a test should be performed to confirm that the module handles the device's absence/unavailability properly.

6) Any other test thought likely to detect an error (based solely on a knowledge of the module's specifications) should be performed; this intuitive strategy is known as error guessing (Ref 32:73-75).

These tests may be combined if they are clearly independent, if their interaction is necessarily a part of a meaningful test, or if equivalence partitioning (Ref 32:44-50) is being used to reduce the number of test cases needed. Other black-box testing methods are certainly possible, such as simple random data entry, and may be performed in individual cases as warranted (Appendix J documents them). The methods listed are simply some of those with the highest probabilities of detecting errors (Ref 32:36-37).

3.9 -- Design Summary

This chapter presented first (Subsection 3.1) an overview (user's view) of the system. Subsection 3.2 then presented the logical design of the system via Data Flow Diagrams (DFD's) and accompanying descriptions of the system's basic data transformations. Next the physical

design was presented (Subsection 3.3) with structure charts and descriptions of the actual physical modules responsible for carrying out the system's operations. Then in Subsection 3.4 the system's essential hardware components were described and their importance justified. Subsection 3.5 described the system's data acquisition application programs. Next (Subsection 3.6), the system's data processing application programs were described somewhat abstractly through discussions of their basic characteristics accompanied by mathematical statements of their function. In Subsection 3.7, the system's data display application programs were treated. Finally (Subsection 3.8), the system's general testing procedures and criteria were presented as informative forerunners of the complete test plan (with specific testing criteria and test results) found in Appendix J. The next chapter, Chapter 4, now deals with the implementation and testing of this chapter's design.

Chapter 4 -- Implementation and Testing

This chapter describes the system's implementation and test results. Section 4.1 presents difficulties that were encountered with the system at the beginning of the implementation and testing phase. Section 4.2 takes a broad look at some of the first and most basic implementation decisions and discoveries. Section 4.3 portrays the User Interface's implementation and testing, with emphasis on several module groupings. Section 4.4 then describes the implementation and testing of the system's application programs, with one subsection for each of those implemented. Finally, Section 4.5 summarizes the implementation results. Note that this chapter contains test summaries; complete specifications of test conditions and results are found in Appendix J.

4.1 -- Initial System Difficulties

The start of system implementation was delayed because of equipment failures suffered by the HP21MX computer and some of its peripherals (the computer was down for a total of nearly two months). Relating the lessons learned while restoring the system to service may help in correcting or preventing similar problems in the future. First, problems with the system's disk drive (which prevented the system from starting up properly) were blamed on the disk controller. Several of the controller's cards were replaced. Later the same problem resurfaced, and the cable connecting the disk

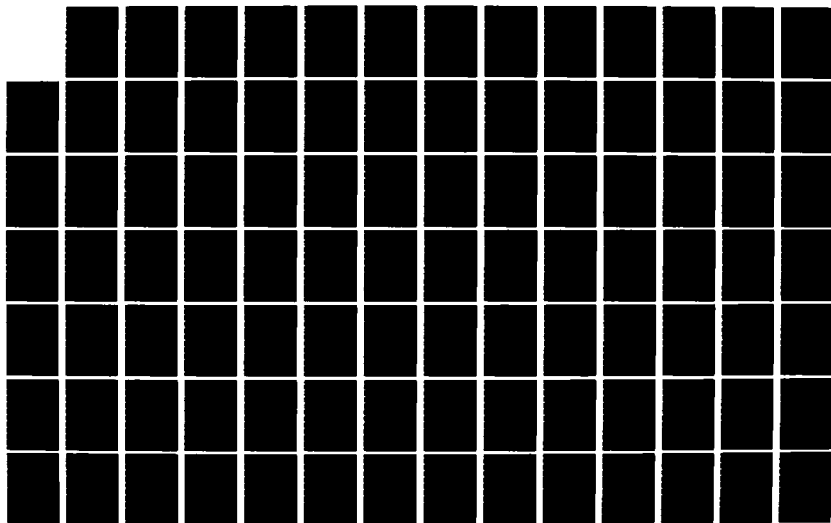
HD-A138 232

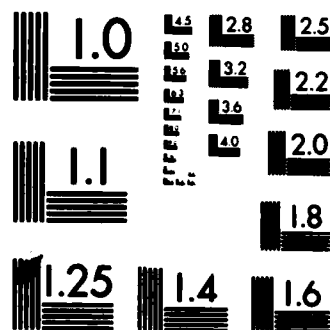
DEVELOPMENT OF A REAL-TIME GENERAL-PURPOSE DIGITAL
SIGNAL PROCESSING LABO. (U) AIR FORCE\$ INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI. J W BENGTON
DEC 83 AFIT/GCS/EE/83D-3 F/G 9/2

2/4

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

drive to its controller was found to be bad (and was probably the cause of trouble all along). This supports the experience of other HP21MX users (Appendix D): interface cables and other sorts of connectors are often at fault when problems occur and should be among the first things checked. With the cable fixed, one other major problem remained: the system "bombed" (inexplicably stopped running) every time a system generation was attempted. System generations must be run each time a new piece of hardware is installed, any changes are made to I/O assignments, etc. For the purposes of this thesis effort, system generations were expected to be needed to install a new tape drive and to install new software (I/O drivers) in support of A/D converters, D/A converters, and the HP1310 terminal. This problem has existed for the recorded history of AFIT's HP21MX system, and has caused tremendous difficulty in previous thesis efforts (Ref 51). The problem was traced to a set of faulty memory boards. Once the boards were removed and the system reconfigured, the system's erratic behavior ended.

4.2 -- General System Characteristics

With a properly functioning system, the implementation began with setting up file accounting and control procedures. A number of removable disk packs (currently six) are available for the HP21MX's HP7906 hard disk drive. They are equal in capacity to the 7906's fixed platter: each holds approximately 10 megabytes. With that large a capacity, the

entire DSP system is easily kept on a single removable disk pack, PACK1 (the packs are numbered sequentially). Every file on the pack is documented in the file name glossary found in Appendix B. Before PACK1 and the other removable packs could be reinitialized for use, it was necessary to break the system's password protection (no one could remember the password). This was done with the FORTRAN program CODE. The system now has no password. While a password can easily be assigned, it obviously doesn't offer much protection anyway and can be the cause of great inconvenience!

Most of the programs described in Section 3.4 came from an IEEE publication on Digital Signal Processing (Ref 13). They were brought onto the DSP system by means of a tape (containing all of the book's programs) acquired from the IEEE. Reading the IEEE tape posed a problem, however: it came in 1600 bpi, blocked, EBCDIC format. The HP21MX's tape drive is 800 bpi, ASCII, and can only read unblocked data, unless user programs are written to perform the block conversions (the inability of the HP's operating system utilities to support blocking is not documented in HP literature; no mention seems to be made anywhere of tape drive formatting specifics). The tape was first converted to the proper format using AFIT's Scientific Support Computer (SSC) and its UNIX utility, DD. All of Reference 13's programs, not just those used by this system, are now on PACK1 and available for use.

Several Operating System (OS) utilities common to more modern systems are not present or behave unsatisfactorily on the HP21MX under its RTE-III OS. First, the system's standard utility for listing files to the printer (LIST) truncates all lines to 66 characters. A FORTRAN program (LST) was written which list up to 132 characters per line (the maximum width supported by the printer), numbers pages if requested, and maintains top and bottom of page margins. RTE-III provides no easy method for gathering together data from a collection of files, such as might be needed to print out a group of source files en masse. To solve this problem a FORTRAN program was written (CAT) which concatenates a set of files into a single file.

It was quickly discovered that memory space limitations would have a major impact on this system's implementation. As mentioned earlier, the maximum size of a user program is limited to 17K words under RTE-III. With only the DSP, GREQ, GOPT, and DMENU modules implemented (together with some service subroutines), the User Interface requires 11K words. Thus, more use is made of disk files for data storage and transfer than had been anticipated (more detail is given in the following sections).

Several languages are available on the HP21MX: ALGOL, BASIC, and FORTRAN compilers/interpreters exist. Since the author doesn't know ALGOL, its use as the system's implementation language was ruled out. BASIC offers a significant advantage over FORTRAN in the area of string

handling -- a major advantage given the large amount of character manipulation done by the User Interface. Nonetheless, FORTRAN use is a necessity, since all of the DSP program from Reference 13 are in FORTRAN and FORTRAN is the only available language to support complex and double-precision data types. The choice was thus between exclusive use of FORTRAN and a combination of FORTRAN and BASIC, with BASIC used to implement the User Interface and FORTRAN used for the DSP application programs. The decision was made to use FORTRAN exclusively for two reasons: first, having two languages rather than one would make the job of system maintenance more difficult, since the programmer would need to be conversant in the peculiarities of HP versions of two languages rather than just one. Second, HP's BASIC is an interpretive version, which means that the interpreter resides in memory along with the program being run -- further restricting available user memory.

4.3 -- User Interface Implementation

The left and right halves of the User Interface (Figure 12) were implemented in sequence, each as a "top-down" implementation (Appendix E). This was possible because of the almost complete independence of the two halves, and desirable because the left half -- which contains all the modules responsible for communicating with the user -- had to be at least minimally operational before decisions could be confidently made concerning the application programs'

communication and control structure. (And of course, those decisions had much to do with determining the types of changes to be made to the application programs in adapting them to this system.)

4.3.1 -- Basic User Interface Communication Modules

The first User Interface modules to be implemented were DSP, GREQ, GOPT, DMENU, BORDR, PREC1 through PREC6, AOPT, and SIOPT. Figures 12 through 15 show that these modules form the "core" of the user-communication side of the User Interface. The top-down dictum of system implementation (Appendix E) was violated in this instance, with all of the modules in this group implemented before others at higher levels were implemented. This was done for several reasons; chief among them was the desire to prove the system's basic design elements workable early in the implementation process. In particular, the ability to communicate with the terminal using screen menus and block data transfers was tested. It was necessary to access the RTE-III driver for the terminal directly to accomplish block transfers and writes to the terminal's screen without carriage return/line feed sequences being inserted at the end of each line (Ref 40:2.5) -- but all of the design aims were proved realizable. Implementing the modules early also made testing of the User Interface's other modules easier and more realistic by allowing them to operate on real, interactively entered data.

It was in implementing these modules that the space problem mentioned in Section 4.1 first caused a change in implementation direction. The fastest method of passing parameters from one module to another is in memory. Unfortunately, many of the data items being passed among the modules in this group are rather lengthy. Each menu page of user input data can easily require hundreds of words of memory space. A few of these data items, combined with only the small percentage of total User Interface code this group of modules represents, left only 6K words of memory remaining out of the total of 17K available. To alleviate this problem (especially to allow more space for present and future editing modules), it was decided to store as many large data items in disk files as possible. Current data values for screen menus and complete user requests are among the items now kept on disk. While response time is somewhat slower, the time required to switch menu pages (the most common of User Interface activities) is still on the order of only a second or two, depending upon the amount of data to be written to the screen. The speed at which the computer communicates with the terminal (9600 baud maximum) is really a more significant limiting factor.

Only two of the modules in this group required code that wasn't completely straightforward: DMENU and AOPT. DMENU makes direct calls to the I/O driver mentioned earlier so that data can be written without being followed by a carriage return/line feed sequence (carriage control features

described in the FORTRAN manual (Ref 41:8.29) don't work on terminal devices). AOPT also uses the device driver directly to accomplish block reads and writes. One of the benefits of making calls directly to the device driver is that it makes use of the HP2648's programmable function keys possible in block mode (Ref 24:5.9,C-11, 40:2.4-6). This allows all User Interface immediate requests to be assigned to function keys, so that the user need only press one key for immediate attention rather than enter an option code from the keyboard and send it along with all the other screen data to the computer for decoding. Consistency is obviously much easier to achieve in this fashion (there is no need to worry about always including certain fields at the same place in each menu display).

Several utility subroutines were written to make the job of dealing with character data in FORTRAN easier; they are among the modules appearing in Appendix H. SESC substitutes the escape character for all capital E's, without regard for which byte (half) of a word the E is in. This is very useful for creating the escape code sequences needed to control the HP2648A terminal: instead of having to include a variable field in a FORMAT statement for the escape character (which can't be represented directly in a FORMAT statement as printing characters can), the character E can be used to represent the escape character. Thus, the user can "see" what the control string looks like directly (with E's representing escapes) and, with a simple call, transform the

string into its final operational form. A generalization of SESC is SCHAR, which allows a character to be substituted for another arbitrary character, again without regard for which byte the character is in. (SCHAR does require that both characters be specified, so SESC is a little easier to use with the cases it was designed for).

Test results for these and all other implemented modules are found in Appendix J.

4.3.2 -- Other GREQ Subordinate Modules

The major modules remaining on the left side of the User Interface are EOPT and FREQ. EOPT (Figure 16) and its subordinates (EM001, EM002,...) consist almost entirely of character manipulation logic, with one EM(NNN) editing routine devoted to each screen menu. No difficulties were encountered in implementing EOPT and its subroutines other than the normal resistance FORTRAN offers to any attempts at character manipulation. FREQ (Figure 17) and its subordinates (FA001, ..., FP001, ..., FD001, ..., CCONS) were similarly straightforward, except that the 17K memory limit was reached in attempting to implement CCONS; thus no consistency checking is done by the current User Interface (an easily correctable situation once the RTE-IV operating system -- with its expanded address space -- is made available). Note that for both EOPT and FREQ subordinate modules (EM(NNN), etc.) were created only for the menus

existing at this time; with each added application program new editing and formatting modules must be added.

4.3.3 -- SAREQ Module

The SAREQ module is where the User Interface and application programs meet. Implementing it was mostly an exercise in developing efficient and effective communication and control procedures. SAREQ communicates with the application programs it executes through system EXEC calls (Ref 39:1.10); it controls them by sending messages to initiate and terminate their actions. SAREQ was relieved of some of its responsibilities because of the memory crunch described earlier (Section 4.1). With all user requests stored in disk files (to save memory), SAREQ has no parameters to pass except file pointers and communication identifiers (Figure 12).

4.4 -- Application Program Implementation

Each of the following subsections covers the implementation and testing of one of the system's application programs (data acquisition, processing, and display, in order). Not all of the application programs were successfully implemented; some for lack of time, some because of memory limitations, and others for lack of hardware or software support from the HP21MX computer. RTE-IV provides enough user-addressable memory to support all those not implemented because of RTE-III memory limitations. While RTE-III provides a 17K-word address space, RTE-IV provides about

28K words (this varies depending upon the system characteristics specified during system generation). For those not implemented, any specific recommendations for their implementation are made in Chapter 5. Source listings of the programs are given in Appendix I. Complete test conditions and results are given in Appendix J.

4.4.1 -- AD001 (Generate Data)

AD001 was successfully implemented and tested. All of its features described in Subsection 3.5.2 are operational, including its ability to interact with concurrent data processing application programs.

4.4.2 -- AD002 (Sample A/D Converter)

AD002 was not implemented, because of time constraints. AD002 is one of two application programs (the other is DD002) requiring that assembly-language I/O drivers be written for communication with external devices -- in this case, with an A/D converter. In addition to the need for an I/O driver, AD002 requires some sort of real-time clocking device; this is discussed in Chapter 5.

4.4.3 -- PD001 (Correlation and Covariance)

PD001 was not implemented because of time constraints and the fact that no more User Interface program space remained for needed editing and formatting modules. However, the program (Ref 13:2.2) was modified so that it would compile and run cleanly on the HP21MX, demonstrating that

its size (12K words) will allow it to be used when RTE-IV is installed.

4.4.4 -- PD002 (Coherence)

PD002 was not implemented. As now configured, the memory address space available to the user on the HP21MX under RTE-III (17K words) is insufficient to allow loading of this program, which requires approximately 23K. Installation of the RTE-IV operating system on the HP21MX should allow PD002 to be used.

4.4.5 -- PD003 (Convolution)

PD003 was not implemented because of the limited address space RTE-III provides. PD003 requires approximately 26K words.

4.4.6 -- PD004 (FFT)

PD004 was successfully implemented according to the design presented in Subsection 3.6.4, and successfully passed its tests as defined in Appendix J. PD004 is able to handle up to 4096 data points per block under RTE-III (memory address space being the limiting factor); it should be possible to double this under RTE-IV.

PD004 was the first data processing application program to be implemented. Because of this, special care was taken to ensure that its documentation was explicit enough to allow its use as an example for future application program

development. In particular, communication protocols are well documented.

4.4.7 -- PD005 (FIR Filter Design)

PD005 was not implemented because of RTE-III's limited address space. PD005 needs about 18K words of memory, placing it just outside RTE-III's maximum of 17K. It should easily fit within RTE-IV's address space.

4.4.8 -- PD006 (IFFT)

PD006 was successfully implemented according to the design presented in Subsection 3.6.6, and successfully passed its tests as defined in Appendix J. PD006 was the second application program to be implemented, providing an easy-to-check complementary program pair with PD004. As with PD004, PD006 handles up to 4096 data points per block under RTE-III, with twice that many likely to be possible under RTE-IV.

4.4.9 -- PD007 (IIR Filter Design)

PD007 would not fit into the 17K words of user-addressable memory available under RTE-III, and so was not implemented. PD007 should fit in RTE-IV's address space.

4.4.10 -- PD008 (Waveform Averaging)

PD008 was not coded or implemented because of time constraints.

4.4.11 -- DD001 (Display to HP2648A)

DD001 was successfully implemented according to the

design presented in Subsection 3.7.1. The HP2648A's graphics abilities made DD001's implementation a fairly simple process. The HP2648A automatically produces graphs when provided with a short set of parameters and a collection of coordinate-value pairs ("AUTOPLOT" (Ref 24:3.24-31)).

Testing of DD001 was uneventful. All Appendix J tests were successfully passed.

4.4.12 -- DD002 (Display to HP1310)

DD002 was not implemented because of time constraints. Since DD002 is to operate the HP1310 graphics terminal directly, much of the work required to implement DD002 is involved with writing an assembly-language driver for the D/A converter responsible for communicating with the HP1310 (not a trivial matter) (Ref 42).

4.5 -- Implementation Summary

This chapter presented the system's implementation. As implemented, the system provides both a set of basic, useful, easy-to-use DSP services, and the groundwork for continued development of hardware and software elements. The next chapter (Chapter 5) details the conclusions reached in this thesis effort and makes recommendations for further development of the HP21MX DSP system.

Chapter 5 -- Conclusions and Recommendations

This thesis effort demonstrates that the HP21MX computer system is well suited for its role as a general-purpose DSP system, with further hardware additions required only to support its real-time abilities. The requirements and design presented in this thesis have been proven workable, and the resulting system's performance gives promise of its long-term usefulness. In particular, the following main conclusions may be stated (compare with the requirements given in Section 1.2):

- 1) The system is capable of supporting the most common of DSP applications. Several of them (data generation, FFT, IFFT, graphics display to the HP2648A terminal) are now operational as a result of this thesis effort, and all others identified in Section 2.3 (with the exception of waveform averaging) are available on the system from Reference 13, needing only to be integrated into the system.

- 2) Because of the modular design of the system, its complete documentation, and help in the form of the System Manager's Guide, system maintenance and further system development are well-defined, well-ordered processes. Unfortunately, this doesn't mean that they are easy processes. For example, HP's FORTRAN offers none of the extensions that are common on newer versions of FORTRAN. Most significantly, HP's FORTRAN provides no help in manipulating character-type information. Without CHARACTER data types, string processing subroutines, etc. nearly every

character manipulation task turns into an exercise in ANDing and ORing words to allow their individual byte/character components to be examined. Several user-written subroutines (e.g. SCHAR, SESC) are included in the DSP system to help with some of these problems; but they don't obviate the fact that the system manager will have to be very well-versed in HP's FORTRAN and its character manipulation characteristics.

3) The system's greatest success was achieved in making it easy to use. Screen menu prompts, user-friendly error-trapping and recovery, on-line help, and a "Plain English" User's Guide all play a part in getting a new user to the point of doing real work much more quickly than would be expected with a less friendly system.

These steps are recommended for the system's further development:

1) Implement the data acquisition application program AD002, which samples the system's A/D converter. An assembly-language I/O driver will need to be written, installed in the system, and interfaced to AD002 (all difficult tasks).

2) Finish implementing the data processing application programs given in Chapter 3. With two completed programs to consult as examples, this should pose no insurmountable problems for an implementer well-versed in HP's FORTRAN.

3) Implement the data display application program DD002, which drives the HP1310 graphics terminal. As with AD002, this means writing an assembly-language I/O driver.

It also means interfacing the HP's D/A converter to the terminal, which may or may not be an easy task.

4) Finish implementing the remaining portions of the User Interface. These modules are not part of the current implementation (they wouldn't fit within the 17K memory limit): CCONS, DIREC, REREQ, and SVREQ. What this means is that, at present: no consistency checking of user requests is performed; no directory listings are provided from within the User Interface; collected sets of user requests may not be saved (the system's "submit file" facility); and collected user requests may not be recalled. It is possible to save one collected request (i.e., a set of all default values associated with all of the system's menus) by making the collected request values the default values (default value saving and recalling is supported by special function keys).

5) Solve the problems involved in controlling the system's A/D converter so that its sampling rate can be interactively specified by the user just as any other parameter can. The HP21MX has a time base generator (TBG) which should make this possible. The TBG can be programmed to provide interrupts at regular intervals (any decade multiple of 0.1ms). Note that this means the maximum sampling rate possible using the TBG as an interrupt source is 10kHz. The TBG must be combined with assembly-language software to handle the TBG's interrupts by activating the A/D converter's I/O driver when the interrupts occur (Ref 21:3.2). Unfortunately, the RTE operating system uses the

TBG for some of its own purposes, most obviously for the maintenance of real-time date and time of day information. While the TBG does support multiple concurrent timing requests (even though it has only one channel), an investigation of RTE's use of the TBG needs to be conducted to ensure that RTE permits conflict-free use of the TBG by application programs (Ref 21). One possible alternative is to clock the A/D converter with an external source (some sort of signal generator) which the user can control directly. This would get around the TBG problems just mentioned, but would require the user to have knowledge of the operation of the signal source (outside the software control of the DSP system).

6) Make use of AFIT's second HP21MX computer (now sitting idle) as a co-processor. Without some kind of multi-processor support, this thesis' system will likely never be capable of useful real-time processing. The reason has to do with the number of independent tasks that must be accomplished concurrently by the system -- acquisition, processing, and display, with each task perhaps having subtasks (calls to I/O drivers, etc.) A half-dozen or more processes could easily be required to execute concurrently. The second HP21MX computer could be used to handle a part of the workload, such as all A/D and D/A data transfers to/from memory or disk.

Bibliography

1. Anderson, Brian D. and John B. Moore. Optimal Filtering. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979.
2. Batch-Spool Monitor Reference Manual. Hewlett-Packard Company, May 1979.
3. Blinchikoff, Herman J. and Zverev, Anatol I. Filtering in the Time and Frequency Domains. New York: John Wiley & Sons, 1976.
4. Bergland, Glenn D. "A Guided Tour of the Fast Fourier Transform," IEEE Spectrum, 6 (7): 41-52 (July 1969).
5. Bergland, Glenn D. and Gordon, Ronald D. Tutorial: Software Design Strategies. New York: IEEE Computer Society, 1981.
6. Boehm, Barry W., et al. Characteristics of Software Quality. New York: North-Holland Publishing Company, Inc., 1978.
7. Brignell, John and Rhodes, Godfrey. Laboratory On-Line Computing. London: International Textbook Company Limited, 1975.
8. Brown, P. J. "Error Messages: The Neglected Area of the Man/Machine Interface?," Communications of the ACM, 26 (4): 246-249 (April 1983).
9. Cappellini, Vito. Digital Filters and Their Applications. New York: Academic Press, 1978.
10. Chamberlin, Hal. Musical Applications of Micro-processors. New Jersey: Hayden Book Company, Inc., 1980.
11. Chen, Chi-Tsong. One-Dimensional Digital Signal Processing. New York: Marcel Dekker, Inc., 1979.
12. Childers, Donald G. and Allen E. Durling. Digital Filtering and Signal Processing. San Francisco: West Publishing Company, 1975.
13. Cooley, J.W., et al. Programs for Digital Signal Processing. New York: John Wiley and Sons, Inc., 1979.
14. Dehning, Waltraud, et al. The Adaptation of Virtual Man-Computer Interfaces to User Requirements in Dialogs. New York: Springer-Verlag, 1981.

15. DeMarco, Tom. Structured Analysis and System Specification. New York: Yourdon, Inc., 1978.
16. DOS/RTE Relocatable Library Reference Manual. Hewlett-Packard Company, December 1978.
17. Foley, James and Andries Van Dam. Fundamentals of Interactive Computer Graphics. Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.
18. Freeman, Herbert, editor. Tutorial and Selected Readings in Interactive Computer Graphics. IEEE Computer Society, 1980.
19. Hamilton, M. and Zeldin, S. "Higher Order Software -- A Methodology for Defining Software," IEEE Transactions on Software Engineering, 2 (1): 9-31 (March 1976).
20. Hodge, Theo. W. and Anita P. Skelton. "A General Purpose Mini-Computer Based Digital Signal Processing Laboratory," NRL Memorandum Report 3502 (May 1977).
21. HP 12539C Time Base Generator Interface Kit Operating and Service Manual. Hewlett-Packard Company, June 1978.
22. HP 12555B Digital-to-Analog Converter Interface Kit Operating and Service Manual. Hewlett-Packard Company, March 1973.
23. HP 21MX Computer Series Operator's Manual. Hewlett-Packard Company, July 1974.
24. HP 2648A Graphics Terminal Reference Manual. Hewlett-Packard Company, August 1979.
25. Kernighan, Brian W. and Plauger, P.J. The Elements of Programming Style. New York: McGraw-Hill Book Company, 1978.
26. Ledgard, Henry F. "The Case for Structured Programming," Bit, 13: 45-57 (1973).
27. Ledgard, Henry F., et al. Directions in Human Factors for Interactive Systems. New York: Springer-Verlag, 1981.
28. Lynn, Paul. An Introduction to the Analysis and Processing of Signals. New York: Halsted Press, 1973.
29. Madnick, Stuart E. and John J. Donovan. Operating Systems. New York: McGraw-Hill Book Company, 1974.

30. Martin, James. Design of Man-Computer Dialogues. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1973.
31. Mehlmann, Marilyn. When People Use Computers. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
32. Myers, Glenford J. The Art of Software Testing. New York: John Wiley & Sons, 1979.
33. "National Bureau of Standards Programming Environment Workshop Report," ACM Software Engineering Notes, 6 (4): (August 1981).
34. Newman, William and Robert Sproull. Principles of Interactive Computer Graphics. (Second Edition) New York: McGraw-Hill Book Company, 1979.
35. Oppenheim, Alan V. Applications of Digital Signal Processing. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.
36. Oppenheim, Vlan V. and Schafer, Ronald W. Digital Signal Processing. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.
37. Papoulis, Athanasios. Signal Analysis. New York: McGraw-Hill, Inc., 1977.
38. Rabiner, L.R. and B. Gold. Theory and Application of Digital Signal Processing. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.
39. Real-Time Executive III Software System Programming and Operating Manual. Hewlett Packard Company, October 1975.
40. RTE Drivers DVR05/DVA05 for HP 263X/264X Terminals. Hewlett Packard Company, July 1979.
41. RTE FORTRAN IV Reference Manual. Hewlett-Packard Company, July 1979.
42. RTE Operating System Driver Writing Manual. Hewlett-Packard Company, January 1980.
43. RTE-IV Programmer's Reference Manual. Hewlett-Packard Company, January 1980.
44. Schoon, Gary A. Applications Directed Microprogramming on a Minicomputer System, AFIT Thesis, (December 1982).

45. Shneiderman, Ben. "Designing Computer System Messages," Communications of the ACM, 25 (9): 610-611 (September 1982).
46. Smith, H.T., et al. Human Interaction with Computers. New York: Academic Press, 1980.
47. Stanley, William D. Digital Signal Processing. Reston, VA: Reston Publishing Company, Inc., 1975.
48. Steidle, John J. Microprogramming: A Tool to Improve Program Performance, AFIT Thesis, (December 1981).
49. Taylor, F. and S. Smith. Digital Signal Processing in FORTRAN. Lexington Books, 1976.
50. Time Series Systems Operating Manual. Time/Data Corporation, 1973.
51. Todd, Wayne. A General Purpose Mini-Computer Based Digital Signal Processing Laboratory, AFIT Thesis, (June 1982).
52. Wasserman, Anthony J. "User Software Engineering and the Design of Interactive Systems," Proceedings of the Fifth International Conference on Software Engineering, IEEE Computer Society, 1981.
53. Weinberg, Victor. Structured Analysis. New York: Yourdon Press, 1978.
54. Yourdon, Edward and L. L. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Englewood Cliffs, N.J.: Prentice-Hall, 1979.
55. Zohar, Stalhav. "Fast Hardware Fourier Transformation through Counting," IEEE Transactions on Computers, C-22 (5): 433-441 (May 1973).
56. 21MX M-Series and E-Series Computer I/O Interfacing Guide. Hewlett-Packard Company, July 1977.

APPENDIX A User's Guide

Introduction

This User's Guide describes the AFIT HP21MX Digital Signal Processing (DSP) System (Ref 1). It is meant to serve both as an initial introduction to the DSP System and as a reference source for certain problems lying outside the scope of its on-line help facilities (such as how to get the system running). The guide is divided into four sections presenting: (I) An overview of the system, (II) How to get the system running, (III) How to use the system's DSP functions, and (IV) How to stop the system when finished. A small bibliography follows Section IV.

I. System Overview

The AFIT HP21MX DSP System is a real-time general-purpose system whose primary aim is to make useful DSP capabilities available without the user having to know much about computer operation or principles. Toward that end, the system insulates its users from nearly all details of the HP RTE-III operating system (a large collection of support programs (Ref 2)) and from having to worry about how to make various DSP programs work with each other. All interaction with the system is handled by a collection of software known as the User Interface. Once the system is running, the user doesn't have to be concerned about any of the low-level details of the system's operations. Instead, the user simply

follows the User Interface's prompting (via menus) and specifies the processing desired on a functional level. The next section deals with what it takes to get the system running.

II. Getting the System Running

This is one area where the HP21MX computer clearly shows its age. Unlike most more modern computers, the HP21MX requires that its user play an integral part in even the very earliest stages of getting the system running.

The first step is, of course, to turn everything on. The system manager may have everything controlled by a single switch, or the system's various components may need to be powered-up individually (if so, get the system manager to show you where the switches are; some are in the rear of the system's chassis, and others are covered by panels). When all the components have power, one additional switch needs to be turned on. It's found on the front of the disk drive unit, marked "RUN/STOP", and controls whether the disk inside is spinning or not (turn it to "RUN"). Once all the components have power and the disk drive is starting to spin up, the computer needs some instructions.

All computers have some sort of an initializing program that gets things going, known as a "bootstrap" program. Unfortunately, the HP21MX doesn't know where its bootstrap program is without being told (modern computers usually do know). Therefore, the first instructions given to the

computer have to do with where it can find its bootstrap program. The program's location is specified by entering a memory address on the computer's front panel. Follow these steps:

1. Turn the computer's on-off key all the way counter-clockwise and then back to the "OPERATE" position. This initializes certain internal components and should cause some of the front panel lights to come on.

2. Use the rocker switch marked located mid-panel (with arrows on either side) to move the red light closest to it so that it's at the "S" position. This selects the proper CPU register to receive the program location.

3. The location is given to the computer in binary by setting certain of the numbered red lights to on (push up the rocker switches beneath them to turn them on, push down to turn them off). Turn on lights 15, 12, 9, 7, and 1 (i.e., 111202 in binary). All of the lights can be cleared at once by pushing the "INSTR STEP/CLEAR DISPLAY" switch to "CLEAR DISPLAY".

4. Press the "STORE/DISPLAY" rocker switch toward "STORE" so that what's on the computer's front panel is stored for access by the CPU.

5. Press the "PRESET/IBL" rocker switch to "PRESET" and then to "IBL" (Initial Binary Load). This clears the computer's memory and loads the bootstrap program into memory from the memory chip it is permanently kept in.

6. Press the "RUN/HALT" switch toward "RUN".

At this point you should immediately hear noise from the disk drive, if it has had time to come up to speed (shown by the "Drive Ready" message on its front panel). Give it a minute or so if nothing is happening. If nothing happens after about a minute, go back through steps 1 through 6, taking special care with steps 3 and 4.

The computer's terminal should now have a message on its screen. If it doesn't, check to make sure that the terminal's switch settings are correct. All of the terminal's pushdown switches should be up (unlatched) except for the REMOTE and CAPS LOCK switches, which should be down. The terminal should be set for communication at 9600 baud with parity set to "NONE" and duplex set to "FULL" (these switches are located at the upper left hand corner of the keyboard). After confirming/changing their settings, hit return a few times; if the computer still hasn't put anything on the screen, consult the system manager.

III. Using the System

With the system up and running, type RU,DSP and hit return. Use the backspace key to correct any errors. From this point on, you will be assisted in all of your actions by the DSP program, which provides prompts, error correction hints, and general help for every function available on the system. You will find yourself able to write only in certain areas of the terminal's screen; these areas are known as unprotected fields (the areas you are not permitted to write

in are, of course, protected fields). With unprotected fields, the length of each entry is restricted to some maximum number of characters (defined by the creator of the menu). Typing beyond that maximum number of characters will automatically put you into the next unprotected field. To move forward to the next unprotected field, hit the TAB key; to move backward, hit control-TAB. Once all desired changes have been made to the screen's data, press the ENTER key (above the number 9 key) -- it takes the place of the RETURN key when using unprotected fields. The best way to learn the system is to work with it. The only caution you need to show is in destroying existing files which may have importance to other users. It's best to do any saving of data or parameters to new files (the system has a very large amount of file space -- about 20 million characters worth) unless you're certain that an existing file can be uneventfully overwritten.

The only part of the system's operation requiring independent explanation is its ability to provide certain immediate services that are not presented in screen prompts. On the upper right portion of the terminal's keyboard there are eight keys arranged in two rows. Some of the keys have a label underneath. For example, the first key's label is HELP. These keys may be pressed at any time to initiate their stated function. Pressing the HELP key will cause the system to display explanatory text directly related to the current choices being offered to you on the screen. When

finished with the text, you'll find yourself back facing the same choices you did when you hit HELP. Pressing the BACKUP key will take you back to the menu you last saw, eventually leading to the system entry menu. Pressing the SAVE NEW DEFAULTS key will save the values on the screen so that they will always come up as the standard default values (until changed again). If you've changed the values on the screen and want to get back the default values to start from scratch (on that menu), press the READ DEFAULTS key and the system's default values will be read in again (note that this does NOT undo a SAVE NEW DEFAULTS). The last special function key is the EXIT key. Pressing it will cause an immediate exit from the system (and loss of any screen data you have not saved).

In general, putting the system to work means specifying: a data acquisition function, a data processing function, and a data display function. With the system's prompting and the HELP key to guide you, you should find it to be a fairly straightforward process.

IV. Stopping the System

The first step toward stopping the system is exiting the DSP program. This is accomplished by pressing the EXIT function key. Once the DSP program is exited, the first thing that should be done is to turn the disk drive's "RUN/STOP" switch to STOP so that the disk will start to slow down; it should be completely stopped (signified by a DOOR UNLOCKED message on the drive's front panel) before its power

is turned off. If the disk isn't permitted to stop spinning before power is removed, damage to the disk drive may result. With the disk drive stopped, the system's master power switch or the individual component power switches may be turned off, and system shutdown is complete.

User's Guide Bibliography

1. Bengtson, John W. Development of a Real-Time General-Purpose Digital Signal Processing Laboratory System, AFIT Thesis, (December 1983).
2. Real-Time Executive III Software System Programming and Operating Manual. Hewlett-Packard Company, October 1975.

APPENDIX B System Manager's Guide

Introduction

This Guide is meant to serve as an aide to both system maintenance and continued system development by the person designated as the DSP system's manager. Section I is a system overview. Section II contains information to help the system manager become acquainted with Hewlett-Packard (HP) concepts and conventions. Section III deals with adding new functions to the system (i.e., new application programs such as FFT's). The final section, Section IV, covers basic system maintenance operations, and includes examples of some of the more important ones. A short bibliography and appendices containing a list of the system's file names and its documentation are found following the final section.

I. System Overview

From the system manager's point of view, the HP21MX DSP system is a collection of FORTRAN modules executing on the HP21MX computer under the control of a User Interface executive program (all of which is, of course, supported by the RTE-III operating system) working together with a collection of peripherals that make it possible to gather, process, store, and display DSP data.

The system manager's responsibility is to ensure that all of the components of the system continue to work together properly, and to add additional components (hardware and

software) when necessary. Being able to do these things requires a system manager with three basic skills: 1) Fluency in the FORTRAN programming language (Ref 5); 2) Proficiency with HP's Real-Time Executive (RTE) operating system (Ref 1,3,4); and 3) Knowledge of low-level HP computer fundamentals (such as HP assembly-language programming and I/O device interfacing). To help the system manager in becoming comfortable with HP's operation and with the DSP system's reliances on non-standard HP concepts and conventions, the next chapter provides an introduction to the system's basic concepts and conventions.

II. System Concepts and Conventions

The first step to becoming adept with the RTE operating system is gaining an understanding of its two-tiered nature. RTE's developers borrowed heavily from earlier HP operating systems; specifically, some that were around when nothing but paper tape was available for mass storage. The lowest level portion of RTE -- which is called just RTE -- provides only the most basic of services, including very few related to mass storage devices (hard disk drives, in the case of this system). As far as pure RTE is concerned, "files" don't exist; only tracks, sectors, etc. exist. In the course of RTE's evolution, a large OS program known as FMGR was added. FMGR is necessary for almost any useful maintenance or development work done on the system. For example, renaming files or moving them from one disk to another without use of

some auxiliary program is possible only through F. ER. In addition, all FORTRAN I/O calls to disk are handled by accessing FMGR modules. In more modern operating systems, RTE and FMGR equivalents are usually integrated. Their not being integrated here poses no real problems, but does mean that a basic knowledge of the operating system's structure is essential to preventing confusion and to being able to accomplish even trivial tasks at times. Three HP manuals are particularly helpful in presenting RTE/FMGR (Ref 1,3,4).

Several HP concepts are unusual and pervasive enough to warrant special attention. One of the most frequently used unusual HP terms is cartridge (Ref 4:1.14). Cartridge refers to a collection of tracks on a hard disk. Each cartridge is defined during system generation and contains its own directory structure. Thus, the cartridge concept provides a means of enforcing data integrity rules (each cartridge can be assigned an independent password), and also provides a way to control the length of time required for directory searches (by limiting the amount of data under the control of any one cartridge directory). Another HP-specific term is select code (see various hardware manuals). Each external device connected to the HP21MX has some sort of interface board plugged into a slot in the back of the HP21MX. The slots are numbered from 10 to 20 (octal); select codes are nothing more than these slot numbers. For a given select code (I/O device) there can be several logical unit numbers of the sort that are used in FORTRAN reads and writes. For example, the

HP2648A terminal is currently assigned select code 13, with logical unit numbers 1 (screen/keyboard), 4 (left cassette drive), and 5 (right cassette drive) assigned to its somewhat independent components.

By convention, all disk files are prefixed with a specific character as a means of identifying the file's type and usage. These are the characters and their meanings:

```
& -- Source file (ASCII)
% -- Relocatable binary (compiled)
' -- List file (compiler output listings, etc.)
(none) -- Memory-image file (linked and executable)
```

III. Adding New Functions

Adding a new data acquisition, processing, or display module to the system requires that seven types of changes be made to the User Interface. Of course, before changing the User Interface, the new program itself must have communication and control logic added to it; the existing application programs should serve as adequate examples. The only part of designing application programs for this system that should be strange to an experienced FORTRAN programmer is HP's CLASS I/O service. The RTE-IV reference manual (Ref 7) has a good treatment of CLASS I/O -- much better than that given in the RTE-III reference manual (Ref 3). The changes to be made in the user interface are best handled as follows:

- 1) Define the new program's parameter requirements; use those requirements to guide the design of a screen menu for their entry.

2) Create an MD(NNN) file containing the menu's design (see the Data Dictionary for MD(NNN) data file definitions).

3) Enter the menu in the TMENT file so that it will be recognized by the User Interface (again, consult the Data Dictionary).

4) Change the parent menu's edit module logic (an EM(NNN) module), and its TMENT file entries, so that entry into the new menu is permitted.

5) Create an edit module (EM(NNN)) for the new menu and install it in the system, and provide a call for it from the EOPT module.

6) Create a help file (HLP(NNN)) for the menu.

7) Create an FREQ subordinate module to handle formatting the request associated with the new menu, or incorporate the new menu into an existing FREQ module, as appropriate.

Note that in all of the above actions, the "NNN" mentioned is some unique number/group of numbers assigned to the new program's menu(s). Numbers may be assigned to menus arbitrarily, so long as they are unique (make sure they aren't already in use). The process of adding a new function can be confusing; look to some of the existing parts of the system as examples (the software listings are included in Ref 2).

Once all changes have been made, the new User Interface modules and application program are installed into the system via the following steps:

1) Compile the source programs with the command RU,FTN4,&(NAME),-,- (Ref 5:J.5). This command works best (there's some sort of the bug in the compiler) when the list and object files -- %(NAME) and '(NAME) -- have already been created with the CR command. The FTN4 compiler should create these files itself, but it usually aborts if they don't already exist. They may be created by entering CR,'(NAME)::-20:4:100 and CR,%(NAME)::-20:5:100 (Ref 1:2.19). These create commands direct that a type 4 (list) file be created and that a type 5 (relocatable) file be created, both on logical unit 20 (the removable disk pack), and both 100 blocks in length. Note the two different prefixes to (NAME). (NAME) is the five-character name chosen for the file (and usually for the program also).

2) Enter a line like the following (one for each new module) into the command file "LDDSP": MR,%(NAME)::-20 (Ref 1:2.42). This causes the new module to be included in the next step's link and load operation, along with all the other system modules.

3) Execute the LDDSP command file by typing TR,LDDSP (Ref 1:2.59). On its completion, LDDSP will save the new version of DSP for you (if the link and load was successful).

Be very careful about any changes made to existing menu files (e.g. MD001). In particular, changing the field length for a menu item requires that all its associated editing and formatting modules (EM(NNN) and FA/FP/FD(NNN) modules) be checked for field length dependencies. For example, many

menu items are numeric, and are edited by calling the subroutine NUMER. NUMER is passed the length of the item to be checked; if the item is made longer on the menu without the calls to NUMER (from editing modules) being changed to reflect that new length, the item will not be edited properly. As an example, suppose a field was originally defined and edited as a single numeric character. If it was changed to four numeric characters in its MD(NNN) file without a corresponding change being made in the associated EM(NNN) file, the user could enter non-numeric information from the second character position on, and NUMER would still classify the item as numeric (since it wasn't told to look beyond the first character), leading EM(NNN) to consider the non-numeric item as being strictly numeric.

IV. System Maintenance

Many system maintenance functions -- such as getting rid of old disk files, etc. -- are quite easy to accomplish with knowledge of a few basic RTE commands (Ref 1,3,4). Three of the most important system maintenance functions, however, are not documented adequately in any single manual. The following paragraphs give examples of: how to perform a disk backup to tape, how to perform a restore from tape to disk, and how to change disk packs (and make the new disk recognizable to the DSP system).

Making a Backup

There is no on-line utility for performing a complete disk to tape backup. The "ST" command allows easy on-line backup of individual files (Ref 1:2.23), but for a comprehensive backup the "DSKUP" off-line utility must be used (there is an on-line version of DSKUP, but we don't have it) (Ref 6). Unfortunately, DSKUP performs only physical (device-oriented) saves and restores; there is no provision for logical (file-oriented) processing. DSKUP is kept on one of the HP2648A's cartridge tapes. To load and execute it, follow these steps:

- 1) Turn on all power to the system. Load the DSKUP tape into the HP2648A's left tape drive, and load a tape onto the reel-to-reel tape unit (make sure the drive is switched on-line and that the tape reel has a write ring in it).

- 2) Follow normal system startup procedures, except instead of entering "111202" into the S register, enter 041300. This tells the system to do its initial load from the terminal's left drive instead of from disk. At this point, the light above the tape unit should start flashing, indicating that the load is in progress. If the light isn't flashing, start again from the beginning.

- 3) When the load is complete, the system will halt (the execute light will go out on the HP21MX's front panel). Enter 000013 into the S register, then enter 000002 into the P register (be sure to press STORE in both cases). These

give the DSKUP program the terminal's select code and the program location at which execution is to begin.

4) Press PRESET and then RUN. At this point the following prompting dialogue should begin. If it doesn't, go back to step 2. Any mistakes made in entering data will likely require that you go back to step 2 (DSKUP catches a few obvious errors, but doesn't allow you to change anything that it has already accepted). User responses are underlined.

```
TASK?  
SA  
MAG TAPE CHANNEL#?  
16  
SOURCE DISC CHANNEL#?  
12  
SOURCE DISC TYPE?  
7906  
SOURCE DISC DRIVE#?  
0  
TYPE OF SAVE?  
FR  
RTE OR DOS DISC?  
RTE  
FROM CYLINDER#?  
0  
# TRACKS?  
822  
# SURFACES?  
2  
STARTING HEAD#?  
0  
6144 WORD BUFFER DESIRED?  
YE  
VERIFY?  
YE  
FILE ID?  
16 Dec 83 822-track FR-TO save of PACK1  
MT FILE#?  
1
```

If everything has been done properly, the reel-to-reel tape unit should start spinning. DSKUP provides messages

telling what it's doing; after about six or seven minutes it will display a "VERIFYING" message and the tape will automatically rewind for the verification procedure. When everything is done, simply remove the newly-created backup tape; the system may then be brought up in its normal fashion. Note that the above procedure is for backing up the removable disk platter. To backup the fixed disk (which contains the operating system, etc.) follow exactly the same procedure except respond to "STARTING HEAD#?" with 2 instead of 0.

Restoring from Tape

Restoring from a backup tape is handled very much like the original backup procedure. Follow the steps given above for executing DSKUP (making sure that the backup tape doesn't have a write ring in it, in this case). Once DSKUP is executing, conduct the following dialogue (user responses underlined):

```
TASK?  
RE  
MAG TAPE CHANNEL#?  
16  
MT FILE#?  
1  
FILE ID:  
  (System prints file ID entered during backup)  
TAPE#: 01  
OK?  
YE (if it really is the right tape)  
DEST DISC CHANNEL#?  
12  
DEST DISC DRIVE#?  
0  
TO CYLINDER?  
0  
# OF SURFACES?  
2
```

STARTING HEAD#?

0 (or 2 for the fixed platter -- BE CAREFUL!)

VERIFY?

YE

DSKUP will now perform the restore. It's a good idea to protect the platter not being restored to using the disk protect switches found under the front panel of the 7906 disk drive. Just make sure that the protection switches are turned off after the restore operation; the system won't operate normally (in fact it won't even boot) with them on. If you want to make absolutely sure that your backup tape contains what you think it does, it's a good idea to try restoring the tape to the scratch disk (PACK4) first. Mount PACK4 (just put it in with the system turned off), restore to it, and convince yourself that what's on it is what you want to have written over the existing contents of your pack. The same kind of previewing can be done for the fixed disk; just restore it to head number 0 on the scratch pack.

Changing Disk Packs

There are six disk packs currently available, labelled PACK1 through PACK6. They are assigned storage duties as follows:

<u>Pack</u> <u>Label</u>	<u>Cart.</u> <u>Number</u>	<u>Contains</u>
PACK1	101	Complete DSP system (software & data)
PACK2	102	RTE-III backup (copied from fixed plat.)
PACK3	103	Steidle/Schoon thesis work
PACK4	104	Scratch pack
PACK5	105	RTE-IV operating system (?...)
PACK6	106	DSP software from ILS system (nonwork.)

Normally, the system will be run using PACK1; it has all of the DSP software, etc. on it. Appendix A contains a list of all the files found on PACK1. Many of the files found on the other packs are unfortunately unidentifiable since no records have been kept by previous users. If a different pack is to be mounted, perform the following actions (it is assumed the system is already running):

- 1) Issue a DC,-20 command (Ref 1:2.88). This tells the system that you are planning on removing the pack found on logical unit 20; i.e., the removable pack. The minus sign tells DC that you are giving it a logical unit number, not a cartridge number. Using "-20" instead of "101", "102", etc. keeps you from having to know and specify which specific pack is actually mounted at the time.

- 2) Turn the disk drive's RUN/STOP switch to STOP. Wait for the drive door open light to come on (signifying that the disk has stopped spinning).

- 3) Open the drive door, remove the old pack, insert the new pack.

- 4) Turn the disk drive's RUN/STOP switch to RUN. Wait for the drive ready light to come on.

- 5) Issue an MC,20 command (Ref 1:2.81). Note that there is no minus sign preceding the 20; this is because the MC command, unlike the DC command, expects that it will be given a logical unit number rather than a cartridge number (in spite of its name -- MC -- which means "Mount Cartridge").

System Manager's Guide Bibliography

1. Batch-Spool Monitor Reference Manual. Hewlett-Packard Company, May 1979.
2. Bengtson, John W. Development of a Real-Time General-Purpose Digital Signal Processing System, AFIT Thesis, (December 1983).
3. Real-Time Executive III Software System Programming and Operating Manual. Hewlett-Packard Company, October 1975.
4. RTE: A Guide for New Users. Hewlett-Packard Company, April 1977.
5. RTE FORTRAN IV Reference Manual. Hewlett-Packard Company, July 1979.
6. RTE Utility Programs Reference Manual. Hewlett-Packard Company, June 1978.
7. RTE-IV Programmer's Reference Manual. Hewlett-Packard Company, January 1980.

System Manager's Guide Appendix A
File Name Glossary (PACK1 Disk Pack)

This glossary contains the names and a description of all files found on the PACK1 removable disk pack. All source files are listed only once, with a "&" preceding them (they may also exist with a "%", an "'", and with no prefix at all). The files are listed alphabetically with prefixes disregarded.

<u>Name</u>	<u>Description</u>
&ALDSP	Collection of all user interface software. Created by the CAT program.
&AOPT	ACCEPT_OPTION user interface subroutine.
&BORDR	DRAW_BORDER user interface subroutine.
&CAT	FORTRAN program to concatenate a group of files into a single file.
&CLRET	CLEAR_ERTEXT user interface subroutine.
&CLRRB	CLEAR_RECBUF user interface subroutine.
&CLRTC	CLEAR_TCONT user interface subroutine.
COMP	Procedure file for compiling and loading a FORTRAN program.
COMPO	Procedure file for compiling a FORTRAN program.
CRE1	Procedure file for creating all of the DSP system's list (') files.
CRE2	Procedure file for creating all of the DSP system's object (%) files.
CRE3	Procedure file for compiling all of the DSP system's modules and saving the compilation results to individual % and ' files.
DICT	This file name glossary.
&DMENU	DISPLAY_MENU user interface subroutine.
&DSP	DIGITAL_SIGNAL_PROCESSING user interface subroutine.
&EOPT	EDIT_OPTIONS user interface subroutine.
&ERMSG	ERROR_MESSAGE user interface subroutine.
&FKEYS	FORTRAN program for defining some of the 2648A's function keys with aids for programming.
FNAMES	File containing names of files to be concatenated by the CAT program.
&FORM	FORTRAN program to test out the 2648's format mode.
&FREQ	FORMAT_REQUEST user interface subroutine.
FRTAPE	Procedure file for reading a tape containing 33 IEEE DSP programs into disk files (SECN.N series).
&GREQ	GET_REQUEST user interface subroutine.
GODSP	Procedure file for putting together all of the user

interface source files, compiling them, and loading them. Calls the CAT program.

&GOPT GET_OPTIONS user interface subroutine.

&GCOM GET_COMMON user interface block data subroutine.

HEADER Outline of documentation header used in all program elements.

LDDSP Procedure file for moving all of the DSP object (%) files into the LG track area, loading them, and storing the resulting executable file as "DSP".

'LIST Shared-use FORTRAN compiler listing file.

&LST FORTRAN program for listing ASCII files. Prints up to 132 characters per line, as opposed to a maximum of 66 for the HP LIST utility. Performs page break at beginning of each subroutine.

MD(NNN) MENU_DATA_FILE, containing menu-defining records (including prompting text and defaults).

%OBJ Shared-use FORTRAN compiler object code output file.

&PREC1 PROCESS_RECORD_1 user interface subroutine.

&PREC2 PROCESS_RECORD_2 user interface subroutine.

&PREC3 PROCESS_RECORD_3 user interface subroutine.

&PREC4 PROCESS_RECORD_4 user interface subroutine.

&PREC5 PROCESS_RECORD_5 user interface subroutine.

&PREC6 PROCESS_RECORD_6 user interface subroutine.

PUR1 Procedure file for purging all DSP list (') files.

PUR2 Procedure file for purging all DSP object (%) files.

&SAREQ SATISFY_REQUEST user interface subroutine.

&SCHAR SUBSTITUTE_CHARACTER user interface subroutine.

SECN.N SECTION_N.N listings from the IEEE publication of digital signal processing software.

&SIOPT SATISFY_IMMEDIATE_OPTIONS user interface subroutine.

TEST Shared-use FORTRAN source code file.

TOTAPE Procedure file for storing 33 IEEE DSP files (the SECN.N series) to tape.

Appendix C

Current and Anticipated DSP System Hardware

This appendix contains a list of all independently identifiable components that are now a part of the DSP system or are recommended for future acquisition. The hardware is divided into two main categories: current system, and anticipated additions.

Current System

Computer

- * HP 2108A M-Series computer

Main Memory System

- | | | |
|---|--------------------|--------------------------------------|
| | 2102A 5060-8360 | Memory Controller (two available) |
| | B 1502-22 | |
| * | 2102B 2102-60001 | Memory Controller (ok for RTE-IV) |
| | C 1830 | |
| * | 12747A | 64K Word Standard Performance |
| | | Memory Module |
| * | 9801031 | National Memory Systems 16K-word |
| | | memory (three in use) |
| * | 12892B 12892-60003 | Memory Protect (ok for RTE-IV) |
| | B 1727 | |
| * | 12897B 12897-60003 | Dual Channel Port Controller (DCPC) |
| | C 1649 | (ok for RTE-IV) |
| * | 12731A 12731-60001 | Memory Expansion Module (ok for RTE- |
| | A 1652 | IV, two available) |
| | 2102A 5060-8359 | 8K Word Standard Performance |
| | D 1540-22 | Memory Module (two) |
| | B 1442-22 | |
| | 2102A 5060-8378 | 4K Word Standard Performance |
| | B 1572-22 | Memory Module |
| | 12896-60001 | DMA Board (two; requires 12895 DMA |
| | A 1122-22 | option; old?) |

Microprogramming

- | | |
|---------------------------------|-------------------------------------------------------|
| * 12945A | User Control Store (UCS) for
M-Series computer |
| * 12978A 12908-60006
1436-22 | Writable Control Store (WCS) for
M-Series computer |

General Accessories

- ```
* 12539C 12539-60003 Time Base Generator
* 12992B Disk Loader ROM for 7905/7906/7920
 disk
```

### General Accessories

- \* 12539C 12539-60003 Time Base Generator
- \* 12992B Disk Loader ROM for 7905/7906/7920 disk
- \* 12992C CRT Terminal Loader ROM for 2648A
- \* 13037-80023 Disk Controller
- Al635

### Interfaces

- \* 12966A 12966-60001 Buffered Asynchronous Data Communi-  
D-2305 42L cations Interface (two)
- \* 12555B 12555-60002 Dual D/A Converter (no RTE-III/IV  
A-1138-22 support)
- 12604B 12604-60001 HP Data Source Interface (no RTE-  
C-1020-6 III/IV support)
- \* 12531 12531-60023 Terminal Interface  
B 1606-22  
12531-80025 1204-22  
12531-6001 A-820-6
- \* 12566B 12566-60024 Microcircuit Interface Card  
1148-22 (two; for A/D converters, etc.)
- 02100-60060 Terminator Board (two)
- 1131-22
- 02116-6198 Dual D/A Converter

### Peripherals

- \* 2648A Graphics Terminal
- \* 5060-6282 Paper Tape Reader
- \* 7906 Disk Drive
- \* H-25 Heathkit Printer (dot-matrix)
- \* ADC-12QZ D/A Converter (two)
- \* 7970B Tape Drive (9-track, 800 BPI)

### Anticipated Hardware Acquisitions

#### Computer

- 92852M Hardware Upgrade Package (for  
RTE-IV)
- 12747H 64K Memory Board

---

\* = Item currently installed and usable with system

APPENDIX D  
WPAFB HP21MX Users

This is a list of all known WPAFB HP21MX users. Those marked with an asterisk were particularly knowledgeable and helpful.

| <u>Name</u>     | <u>Phone</u> | <u>Bldg.</u> | <u>Organization</u> |
|-----------------|--------------|--------------|---------------------|
| Bob Ballard     | 52493        | 24C          | AFWAL/FIMN          |
| John Bankovskis | 76177        |              | FTD                 |
| L. T. Drzal     | 52952        | 32           | AFWAL/MLBM          |
| Sgt. Sam Jiles  | 56361        | 622          | AFWAL/AARI          |
| *Brian Kent     | 54465/55076  | Barn         | AFWAL               |
| *Jim Leonard    | 53050        | 23           | AFWAL/AAFR/2        |
| Irvin F. Luke   | 52372        | 18           | AFWAL/POOC/1        |

## Appendix E

### Introduction to Structured Analysis and Design

#### Introduction

This appendix should help to familiarize readers unacquainted with structured analysis and structured design tools and procedures. The development methodology used is a combination of tools and procedures gleaned from a number of sources and collectively espoused by Victor Weinberg (Ref 39). Tom DeMarco's book (Ref 11.5) is a good source for more detailed information on structured analysis, and Yourdon and Constantine's book (Ref 39.5) goes deeper into structured program design.

#### Development Phases

System developments are typically divided into three broad phases: analysis, design, and implementation. This thesis investigation follows that pattern with one slight deviation due to the fact that a current system analysis isn't required (no current unautomated system exists): the analysis phase is segmented, with preliminary analysis results presented in the requirements chapter and further analysis included in the design chapter.

#### Analysis Phase

In this thesis, the analysis phase is documented in section 3.1, "Logical Design of System". The goal of the analysis phase is to develop a logical model of the system to be developed. This is done with the use of two major tools: the data flow diagram (DFD) and the data dictionary. DFD's



show the flow of data through the logical processes of the system. The data flows are documented in a data dictionary. DSD's provide a graphical description of user logical data structure requirements. DFD's are composed of a few basic symbols: circles, representing data transforms; labeled arrows, depicting data flow in and out of transforms; rectangles, acting as data sources and destinations; and labeled bars, representing files of some sort. Figure 3 is a simple DFD using these symbols. Note that DFD's do not show control information.

Data dictionaries are used to keep track of the terminology associated with a system's development, including: data elements, data flows, and processes. A complete and accurate data dictionary is essential for follow-on system development. See Appendices F and G for examples.

### Design Phase

The goal of the design phase is to translate the logical model of the system produced in the analysis phase into a physical model of the system. Structured design is based on a process of successive decomposition: the system is broken into several main functions, each of which is decomposed into subordinate functions, etc. A new tool is introduced to help: the structure chart.

Structure charts are used to define the structure of the physical model of the system just as DFD's are used to portray the logical model of the system. In fact, structure charts may generally be derived from DFD's by any of several

techniques. The structure chart graphically depicts the modular, hierarchical structure of the system and the passing of data between modules. Each module in the system is represented by a box. Labeled arrows are placed on the lines joining the various boxes to indicate data flow. See Figure 12 for an example.

### Implementation Phase

When a system's design is complete, implementation follows. Implementation refers to the coding and testing of the system. The top-down approach applies here: highest-level modules are implemented first, followed by their subordinates. Each level of modules in a structure chart is generally coded and tested completely before the next level is begun. All code is internally documented. A test plan is produced, and includes: test data, expected results, and any performance requirements.

APPENDIX F  
Data Dictionary  
(Data Items)

Data items are listed alphabetically.

### Legend

### Data Characteristics:

|   |                  |                                                              |
|---|------------------|--------------------------------------------------------------|
| A | Alphabetic ASCII | Any printing ASCII character, subject to stated restrictions |
| C | Complex          | Complex floating-point number                                |
| D | Double           | Double-precision floating-point number                       |
| F | Floating-point   | Single-precision floating-point number                       |
| I | Integer          | Single binary word                                           |
| L | Logical          | Single logical value                                         |
| N | Numeric ASCII    | 0-9, decimal point, minus and plus signs                     |

**+ - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - +**

```
ACQIO = * Class I/O number associated with acquisition
 process *
```

Used by: AD000 through AD(NNN), PD001 through PD(NNN),  
SAREQ.

Changed by: SAREQ.

Characteristics: I

Class I/O numbers are assigned by the system when requested via an EXEC call. They are used to uniquely identify Class I/O senders and receivers.

ACQNO = \* Identifying number of data acquisition module  
user has selected \*

**Used by:** DSP, EOPT, FREQ, GREQ, SAREQ.

Changed by: EM002, GREQ.

Characteristics: I

```
0 = File input (AD000)
1 = Generate data (AD001)
2 = Take data from A/D converter (AD002)
```

```
ACQOT = * Name of file to which acquisition process
 is writing *
```

Used by: AD000 through AD(NNN).  
Changed by: AD000 through AD(NNN).  
Characteristics: 6A (3I)

File names of raw, sampled/generated data files are always of the form "SDnnnn", where nnnn is a four-digit number.

"AREQ" = \* Acquisition request file. Contains all the control data needed by an acquisition module to define the operations expected of it \*  
  
= (acquisition module specific)

Used by: AD000 through AD(NNN)  
Changed by: FA000 through FA(NNN)  
Characteristics: Sequential disk file

ATTRI = \* Character which defines display attributes of menu file text and defaults; found in menu file (MD(NNN)) record types 2 and 5.

Used by: PREC2, PREC5  
Changed by: PREC2, PREC5  
Characteristics: A

"I" = Print text in inverse video  
"N" or " " = Print text in normal video

CBEGIN = \* Character position at which some operation (such as a string substitution) should begin \*

Used by: SCHAR.  
Changed by: (modules calling SCHAR)  
Characteristics: I

CEND = \* Character position at which some operation (such as a string substitution) should end \*

Used by: SCHAR.  
Changed by: (modules calling SCHAR)  
Characteristics: I

CFREQ = \* Coherence function frequency values produced to accompany magnitude-squared coherence (MSC) values \*

Used by: COHER  
Changed by: COHER  
Characteristics: (512)\*F

CHAR = \* Temporary storage for a single character from

some word (each HP word contains two  
characters) \*

Used by: EM001.  
Changed by: EM001.  
Characteristics: I

COLMAX = \* Maximum column number for terminal screen \*

Used by: PREC1.  
Changed by: GCOM.  
Characteristics: I

Columns are numbered from 0 to 79 on the HP2648A  
terminal.

COLMIN = \* Minimum column number for terminal screen \*

Used by: PREC1.  
Changed by: GCOM.  
Characteristics: I

COLUMN = \* Column to which screen cursor is to be  
positioned \*

Used by: CURSR.  
Changed by: (modules calling CURSR)  
Characteristics: I

CONDAT = \* Convolved data \*

Used by: PD003.  
Changed by: EM\_\_\_.  
Characteristics: (512)\*F

CONWD1 = \* EXEC call control word number one; used to  
request a terminal-enabled block data read  
from the HP2648A terminal \*

Used by: AOPT, DMENU, HELP.  
Changed by: GCOM.  
Characteristics: I

CONWD2 = \* EXEC call control word number two; used to  
request a transparent write to the terminal \*

Used by: AOPT, BORDR, CURSR, DSP, ERMSG, HELP, PREC2,  
PREC5, PREC6.  
Changed by: GCOM.  
Characteristics: I

CONWD3 = \* EXEC call control word number three; used to  
request a program-enabled block data read from

the terminal \*

Used by: AOPT.  
Changed by: GCOM.  
Characteristics: I

CORPTS = \* Number of correlation points used in estimating a power spectrum \*

Used by: PD001.  
Changed by: EM\_\_\_\_.  
Characteristics: I

CORPTS must be  $\leq$  PTSBLK/2+1 when used by CORR

CORTYP = \* Type of correlation to be performed \*

Used by: PD001.  
Changed by: EM\_\_\_\_.  
Characteristics: I

0 = Auto-correlation  
1 = Cross-correlation  
2 = Auto-covariance  
3 = Cross-covariance

CORVAL = \* Correlation function output values \*

Used by: PD001.  
Changed by: PD001.  
Characteristics: ( $\geq$ FFTSIZ)I

CPOS = \* Character position identifier; used to keep track of which character in a character string is receiving some operation's attention \*

Used by:  
Changed by:  
Characteristics:

CURMEN = \* Unique number identifying current menu \*

Used by: AOPT, BORDR, DMENU, EOPT, GOPT, GREQ, HELP, PREC1 - PREC6, SIOPT, TMENT.  
Changed by: DSP, TMENT.  
Characteristics: I

N = Menu number N is currently being displayed

"CV(NNN)" = \* Current values file containing the current values associated with menu number (NNN) \*

Used by: DMENU, AOPT  
Changed by: AOPT  
Characteristics: Sequential ASCII disk file

Current values files are structured in the same way that menu files (MD(NNN)) are. The only difference is that comment records are left out of CV(NNN) files.

CVDCB = \* Current value file (CV(NNN)) data control block, used when performing system calls for file I/O operations \*

Used by: AOPT, DMENU.  
Changed by: AOPT.  
Characteristics: 144I

CVNAM = \* Array containing the name of the current value (CV(NNN)) file in use \*

Used by: AOPT.  
Changed by: AOPT.  
Characteristics: 6A (3I)

CXSIG = \* Complex X signal (note that the X signal is considered the primary signal, and the Y signal is the secondary signal (if present)) \*

Used by: PD003.  
Changed by: EM\_\_\_\_.  
Characteristics: ( ) \*C

DIRDES = \* Directory destination. Defines where the user wants his requested directory listing to be sent \*

Used by: DIREC.  
Changed by: SIOPT.  
Characteristics: I

0 = Print out directory on terminal screen  
1 = Print out directory on printer

DIRTYP = \* Directory type. Defines what sorts of files the user wants to have appear in his directory listing \*

Used by: DIREC.  
Changed by: SIOPT.  
Characteristics: 10\*I

Each file type is represented by one of the entries in the array. If an array element con-

tains a 1, then the corresponding file type will be included in the directory listing. If the array element contains a 0, the file type will not be included.

| Position | File Type                    |
|----------|------------------------------|
| 1        | CV(NNN) - Current Values     |
| 2        | ID(NNN) - Input Data         |
| 3        | MD(NNN) - Menu Data          |
| 4        | OD(NNN) - Output Data        |
| 5        | PT(NNN) - Parameter Transfer |
| 6        | RF(NNN) - Request File       |

DISIN = \* Name of file from which display module is reading \*

Used by: DD000.  
 Changed by: EM\_\_\_\_.  
 Characteristics: 6A (3I)

Names of files to be used by a display module are always of the form "ODnnnn". "nnnn" is a unique four-digit identifier.

DISIO = \* Class I/O number assigned to display module user has selected \*

Used by: DD000 - DD(NNN), PD001 - PD(NNN).  
 Changed by: SAREQ.  
 Characteristics: I

Class I/O numbers are provided by the system in response to an EXEC request.

DISNO = \* Identifying number of display module user has selected \*

Used by: DDATA, DSP, EOPT, FREQ, GREQ, SAREQ.  
 Changed by: EM004, GREQ.  
 Characteristics: I

0 = File output  
 1 = Display to HP 2648  
 2 = Display to HP 1310

"DREQ" = \* Display request file. Contains all the data needed to define the operations expected of a selected display application program \*

= (display-program dependent)

Used by: DD000 through DD(NNN)



Changed by: FD000 through FD(NNN)  
Characteristics: Sequential disk file

ERTEXT = \* Text to be printed as an error message to  
the user \*

Used by: ERMSG.  
Changed by: (all modules calling ERMSG)  
Characteristics: 160A (80I)

FBACK = \* Flag used to communicate whether or not the  
user wants to back up to the previous menu \*

Used by: GREQ, SIOPT, TMENT.  
Changed by: AOPT, EM001 - EM(NNN), GREQ.  
Characteristics: L

.TRUE. = User wants to back up to previous  
menu.

.FALSE. = User doesn't want to back up to  
previous menu.

FCONS = \* Flag used to communicate whether or not the  
user's requests are consistent with one  
another (and thus whether or not execution  
should be permitted to commence) \*

Used by: CCONS, FREQ, GREQ  
Changed by: CCONS  
Characteristics: L

.TRUE. = User's requests are consistent --  
execute.

.FALSE. = User's requests are inconsistent --  
do not execute.

FDIREC = \* Flag used to communicate whether or not the  
user wants a directory listing \*

Used by: DIREC, EOPT, GREQ, SIOPT.  
Changed by: AOPT, EM001 - EM(NNN).  
Characteristics: L

.TRUE. = User wants a directory listing.

.FALSE. = User doesn't want a directory  
listing.

FERR = \* Flag used to communicate whether or not a  
fatal error has occurred \*

Used by: All modules  
Changed by: All modules  
Characteristics: L

.TRUE. = Fatal error has occurred.  
.FALSE. = Fatal error hasn't occurred.

When a fatal error occurs, the normal procedure is for the system to print out a descriptive error message and then stop execution.

FEXE = \* Flag used to communicate whether or not the user wants to execute the request that has been formed \*

Used by: EOPT, GREQ.  
Changed by: EM001.  
Characteristics: L

.TRUE. = Execute request.  
.FALSE. = Don't execute request.

FEXIT = \* Flag used to communicate whether or not the user wants to exit the system \*

Used by: CURMEN, DSP, GREQ, SIOPT.  
Changed by: AOPT, DSP, EM001 through EM(NNN).  
Characteristics: L

.TRUE. = User wants to exit the system.  
.FALSE. = User doesn't want to exit the system.

FFTSIZ = \* Number of points on which FFT is to be performed \*

Used by: PD006.  
Changed by: EM\_\_\_\_.  
Characteristics: I

FFTSIZ must always be a power of 2.  
FFTSIZ must be  $\geq 2 \times \text{CORPTS} - 1$  for use by CORR.

FFTYPE = \* FIR filter type \*

Used by: PD005.  
Changed by: EM\_\_\_\_.  
Characteristics: I

1 = Multiple passband/stopband  
2 = Differentiator  
3 = Hilbert transformer

FGEDIT = \* Flag specifying whether edited user entries were good (acceptable) \*

Used by: EM001 through EM(NNN), EOPT, GREQ.  
Changed by: EM001 through EM(NNN).  
Characteristics: L

.FALSE. = User's entries are unacceptable  
.TRUE. = User's entries are acceptable

FHELP = \* Flag specifying that the user wants help with  
the current menu \*

Used by: SIOPT.  
Changed by: AOPT, EM001 through EM(NNN).  
Characteristics: L

.FALSE. = User doesn't want help  
.TRUE. = User wants help

FIRST = \* Flag specifying whether PREC1 is being called  
for the first time or not \*

Used by: PREC1.  
Changed by: DMENU.  
Characteristics: L

.FALSE. = Not first time to call PREC1  
.TRUE. = First time to call PREC1

FLDLEN = \* Length of a menu response field as defined  
by a type-4 record in a menu (MD(NNN)) file \*

Used by: PREC4.  
Changed by: AOPT, DMENU, PREC4.  
Characteristics: I

FRDEF = \* Flag specifying whether the user wants to  
re-read the default values for a menu (useful  
if a lot of entry errors have been made and  
the user wants to start again from scratch) \*

Used by: DMENU, GOPT, SIOPT.  
Changed by: AOPT, EM001 through EM(NNN).  
Characteristics: L

.FALSE. = User doesn't want to re-read default  
values.  
.TRUE. = User wants to re-read default values.

FREDIS = \* Flag specifying whether or not the current  
menu should be displayed again or not \*

Used by: GOPT.  
Changed by: AOPT, GOPT.  
Characteristics: L

.TRUE. = Re-display the current menu.  
.FALSE. = Don't re-display the current menu.

FREREQ = \* Flag specifying whether or not the user wants  
to read a previously-created request into the  
menus \*

Used by: EOPT, GREQ.  
Changed by: EM001.  
Characteristics: L

.TRUE. = Read in a request.  
.FALSE. = Don't read in a request.

FREVAL = \* Frequency values produced by program CORR  
as part of its power spectral density compu-  
tations \*

Used by: CORR  
Changed by: CORR  
Characteristics: (FETSIZ/2+1)F

FS = \* Sampling frequency \*

Used by: AD001, AD002.  
Changed by: EM\_\_\_.  
Characteristics: I

FSDEF = \* Flag specifying that the user wants to save  
the menu's current values, making them the  
new default values.

Used by: SIOPT.  
Changed by: AOPT, EM001 through EM(NNN).  
Characteristics: L

.FALSE. = User doesn't want to save new default  
values.  
.TRUE. = User wants to save new default values.

FSIOPT = \* Flag specifying whether or not the SIOPT  
module should be called to satisfy an  
immediate request \*

Used by: AOPT.  
Changed by: AOPT.  
Characteristics: L

.TRUE. = Call SIOPT.  
.FALSE. = Don't call SIOPT.

FSVREQ = \* Flag specifying whether the user wants to

save the requests which have been formed  
for later user \*

Used by: EOPT, GREQ, SIOPT.  
Changed by: EM001 through EM(NNN).  
Characteristics: L

.FALSE. = User doesn't want to save requests.  
.TRUE. = User wants to save requests.

FX = \* Frequency response desired for each band in  
an FIR filter \*

Used by: PD005.  
Changed by: EM\_\_\_\_.  
Characteristics: 10\*F

"GI(NNNN)"= \* Generated input data file containing raw  
generated data \*

Used by: PD(NNN)  
Changed by: AD001  
Characteristics: Sequential disk file containing  
header information followed by sample values.

| Record (in order):    | Format: |
|-----------------------|---------|
| Identifying text      | 80A     |
| Number of data points | I       |
| Sampling frequency    | I       |
| Sample value          | F       |
| (more sample values)  |         |

HLPDCB = \* Data control block used for system I/O calls  
when reading/writing HP(NNNN) files \*

Used by: HELP.  
Changed by: HELP.  
Characteristics: 144I

HLPNAM = \* Array containing the name of the current help  
file being read \*

Used by: HELP.  
Changed by: HELP.  
Characteristics: 6A (3I)

IMPFIL = \* Name of file used to store filter impulse  
response values \*

Used by: PD005.  
Changed by: EM\_\_\_\_.  
Characteristics: 6A (3I)

File name must be of the form "PTNNNN", where "PT" identifies it as a parameter transfer file, and "NNNN" is a four digit number used to uniquely identify the file.

LAGVAL = \* Lag values produced by the correlation program (PD001) while performing correlation and covariance functions \*

Used by: PD001  
Changed by: PD001  
Characteristics: (FFTSIZ/2+1)I

LASTC = \* Pointer to the last character for some string operation \*

Used by:  
Changed by:  
Characteristics:

LCAPE = \* Contains a capital "E" in its left byte \*

Used by: SESC.  
Changed by: SESC.  
Characteristics: I

LEN = \* Length (in words) of data block read from a file as a result of a call to the system subroutine "READF" \*

Used by: (all modules that read/write files)  
Changed by: (all modules that read/write files)  
Characteristics: I

LENCHR = \* Length of a string, in characters \*

Used by: PREC5.  
Changed by: PREC2.  
Characteristics: I

LENWDS = \* Length of string to be operated on, in words \*

Used by: SESC.  
Changed by: SESC.  
Characteristics: I

LFILT = \* Length of filter \*

Used by: PD003.  
Changed by: EM\_\_\_\_.  
Characteristics: I

LFROM = \* Left-byte value for a character (each word contains two characters) \*

Used by: SCHAR.

Changed by: (all modules calling SCHAR)

Characteristics: I

LGRID = \* Grid density for FIR filter design computations \*

Used by: PD005.

Changed by: EM\_\_.

Characteristics: I

Default value is 16.

LMASK = \* Mask used for logically extracting (ANDing out) the left character in a word \*

Used by: DMENU.

Changed by: DMENU.

Characteristics: I

LMASK = 077400B (octal)

LTO = \* Left-byte character value for "TO" value in character substitution \*

Used by: SCHAR.

Changed by: (all modules calling SCHAR)

Characteristics: I

MAXERT = \* Maximum value of subscript which may be used with the ERTEXT array \*

Used by: BORDR, CLRET, ERMSG.

Changed by: GCOM.

Characteristics: I

MAXREC = \* Maximum value of subscript which may be used with the RECBUF array \*

Used by: DMENU, HELP, PREC1, PREC4, PREC5.

Changed by: GCOM.

Characteristics: I

MAXSCR = \* Maximum value of subscript which may be used with the SCRDAT array \*

Used by: AOPT, PARAM.

Changed by: GCOM.

Characteristics: I

MAXTCO = \* Maximum value of subscript which may be used  
with the TCONT array \*

Used by: AOPT, CLRTC, DMENU, DSP, ERMSG, PREC2, PREC5.  
Changed by: GCOM.  
Characteristics: I

"MD(NNN)" = \* Menu data files containing user prompting  
text and default values \*

Used by: DMENU, PREC1 through PREC6  
Changed by: SDEF  
Characteristics: Sequential ASCII disk file

Each menu has one of these menu files associated with it. The "NNN" in the menu file's name uniquely identifies the menu it will be used with, and corresponds with the same number used to make up a part of certain associated software module names (such as "EM(NNN)").

The menu file consists of text lines followed by sets of lines in groups of six. Each of the groups defines a single entry on a menu page, including where the prompting text is to be placed on the page, what its display attributes will be, the length of the response field, what the response field's attributes will be, and the response field's default value. An example is given below.

0,This is a text line; anything at all may  
0,be placed on it as long as the line starts with a  
0,zero and a comma.  
1,2,8 (Cursor is to be positioned on row 2,  
column 8)  
2,I (Text is to be printed in inverse video.  
If normal display desired, put an "N"  
after the comma)  
3,Number of points? (Prompting text)  
4,3 (The user is to be allowed up to three  
characters/digits for a response)  
5,N (The default value, and the user's  
response when entered, are to be printed  
in normal video rather than inverse)  
6,128 (The default value is 128)  
1,5,60 (New group)  
etc.

Each line in the file MUST start with 0-6 followed by a comma. Type 0 lines may be placed only before all other line types for documentation, and have no effect on processing. Line types 1-6 must occur in order, and all six must be present



in each group.

Note that row and column values used in type 1 records assume that row and column numbering starts with 0, not 1.

**MDDCB** = \* Menu data file data control block; used to hold file control information when opening MD(NNN) files for reading/writing \*

Used by: PREC1 through PREC6.

Changed by: AOPT, FA000 through FA(NNN), FD000 through FD(NNN), FP001 through FP(NNN).

Characteristics: 144I

**MDNAM** = \* Array containing the name of the menu data (MD(NNN)) file in use \*

Used by: AOPT, DMENU.

Changed by: AOPT, DMENU.

Characteristics: 6A (3I)

**MENUSE** = \* Array of flag values which specify whether or not a particular menu has been accessed during the current terminal session \*

Used by: DMENU, FREQ subordinates.

Changed by: AOPT.

Characteristics: 100L

**MENUT** = \* File containing values values which define the structure of the user interface's menu tree (which menus follow/precede which) \*

Used by: TMENT

Changed by: (None)

Characteristics: Sequential ASCII file

The file contains pairs of integers, with one pair on each line (separated by commas). The first integer in each pair is a "parent" menu number. The second integer is a "child" menu number. Each parent can have arbitrarily many children (including none), so parent (first integer) entries need not be unique. Each menu number should appear as a child (second integer) entry only once, however, since each child has only one parent. The order of the entries for a given parent is very significant: the children must be listed so that they may be indexed by the user's menu entry. For example, if the user wants the third option on a (parent) menu, and so enters a 3, the child menu associated with that option must be defined by the third MENUT entry for the current parent.

MESG = \* Message buffer \*

Used by: NOCR, SCHAR, SESC.  
Changed by: NOCR, SCHAR, SESC.  
Characteristics: 80A (40I)

MSC = \* Magnitude-squared coherence values \*

Used by: PD002  
Changed by: PD002  
Characteristics: 1024\*F

MTDCB = \* Menu tree file data control block used when  
performing I/O operations to the menu tree  
file.

Used by: TMENT.  
Changed by: TMENT.  
Characteristics: 144I

MTNAM = \* Array containing the menu tree file name \*

Used by: TMENT.  
Changed by: TMENT.  
Characteristics: 6A (3I)

NOCRLF = \* Data value used to suppress the printing of  
carriage return/line feed sequences on the  
terminal screen \*

Used by: AOPT, BORDR, DMENU, DSP, ERMSG, HELP, NOCR,  
PREC2, PREC5.  
Changed by: GCOM.  
Characteristics: I

NUMBLK = \* Number of data blocks to be processed \*

Used by: PD002.  
Changed by: EM\_\_\_\_.  
Characteristics: I

NUMPAR = \* Number of parameters needed to format a  
request \*

Used by: FA001.  
Changed by: FA001.  
Characteristics: I

OD(NNNN) = \* Output data file containing data ready for  
printing, plotting, etc. \*

Used by: Device routines (and OS commands)

Changed by: DD001

Characteristics: Sequential disk file containing  
header information followed by output data.

Record (in order):

Format:

Identifying text

80A

Output device number  
(Data)

I

(Depends on output  
device)

OPTCNT = \* Option count for options being decoded from  
the SCRDAT array \*

Used by: TMENT.

Changed by: TMENT.

Characteristics: I

OPTNUM = \* Number of option to be decoded from the SCRDAT  
array \*

Used by: TMENT.

Changed by: TMENT.

Characteristics: I

PAGEDE = \* Number of page desired \*

Used by: HELP.

Changed by: HELP.

Characteristics: I

PAGENO = \* Page number count \*

Used by: HELP.

Changed by: HELP.

Characteristics: I

PARBUF = \* Parameter buffer, used to transfer parameters/  
options from SCRDAT or data file to using  
module \*

Used by: AOPT, FPAR, TMENT.

Changed by: PARAM.

Characteristics: 80A (40I)

PARCNT = \* Parameter count \*

Used by: FPAR, PARAM.

Changed by: AOPT.

Characteristics: I

PARNUM = \* Parameter number; 1 = first parameter in file,  
2 = second, etc. \*

Used by: FPAR.  
Changed by: FREQ subordinates.  
Characteristics: I

"PD(NNNN)" = \* Processed data file containing data produced  
by a processing module \*

Used by: PD001 through PD008  
Changed by: PD001 through PD008  
Characteristics: Sequential disk file containing  
header information followed by processed data  
values.

| Record (in order): | Format: |
|--------------------|---------|
|--------------------|---------|

|                        |     |
|------------------------|-----|
| Identifying text       | 80A |
| Number of data points  | I   |
| Number of points/block | I   |
| Sampling frequency     | I   |
| Data type              | I   |

0 = Integer  
1 = Single precision floating-point real  
2 = Double precision floating-point real  
3 = Complex (single precision)

|                    |       |
|--------------------|-------|
| Data values        | F/D/C |
| (more data values) |       |

PROIN = \* Name of file from which process module is  
reading \*

Used by: PD001 through PD(NNN).  
Changed by: EM\_\_\_.  
Characteristics: 6A

Names of files to be processed by a process  
module are always of the form "SD(NNNN)" for raw,  
sampled/generated data files, or of the form  
"PD(NNNN)" for previously processes data. In each  
case, (NNNN) is a unique four-digit identifier.

PROIO = \* Class I/O number assigned to data processing  
module \*

Used by: AD000 through AD(NNN), DD000 through DD(NNN),  
PD001 through PD(NNN).  
Changed by: SAREQ.  
Characteristics: 10I

Class I/O numbers are provided by the system  
in response to an EXEC call, and are then used to

uniquely identify Class I/O message senders and receivers.

PRONO = \* Identifying number of data processing module user has selected. If PRONO(3) = 4, it means that the third processing module is to be a Fast Fourier Transform \*

Used by: DSP, EOPT, FREQ, GREQ, SAREQ.

Changed by: EM003, GREQ.

Characteristics: 10I

0 = No processing  
1 = Auto-correlation, cross-correlation, auto-covariance, cross-covariance, power spectral density  
2 = Coherence  
3 = Convolution  
4 = Fast Fourier Transform  
5 = FIR Filter Design  
6 = Inverse Fast Fourier Transform  
7 = IIR Filter Design  
8 = Waveform Average

PROOT = \* Name of file to which process module is writing \*

Used by: PD001 through PD(NNN).

Changed by: EM\_\_.

Characteristics: 6A

Names of files containing data produced by a process module are always of the form "PDnnnn". "nnnn" is a unique four-digit identifier.

"PRQ(NNN)" = \* Processing request file. PRQ001 contains parameters for the processing program PD001, etc. \*

Used by: PD001 through PD(NNN).

Changed by: FREQ subordinates.

Characteristics: Sequential disk file.

PSDVAL = \* Power spectral density function values \*

Used by: PD001

Changed by: PD001

Characteristics: (>=FFTSIZ)F

"PT(NNNN)" = \* Parameter transfer file containing parameters to be transferred from one process module to another \*

Used by: PD005, PD007  
Changed by: PD005, PD007  
Characteristics: Sequential disk file containing  
header information followed by parameters to  
be transferred.

| Record (in order): | Format:             |
|--------------------|---------------------|
| Identifying text   | 80A                 |
| Parameter type     | I                   |
| (?)                |                     |
| Parameters         | (Depends upon type) |

PTSBLK = \* Number of points per block of data \*

Used by: PD001.  
Changed by: EM\_\_\_\_.  
Characteristics: I

Must be a power of two at present.

RECBUF = \* Record buffer used to initially hold data  
read in from disk files \*

Used by: AOPT, DMENU, HELP.  
Changed by: AOPT, DMENU, ERMSG, HELP, PREC1, PREC2,  
PREC4, PREC5, PREC6.  
Characteristics: 80A (40I)

RECTYP = \* Record type. Hold the first character of  
menu file (MD(NNN)) record, which is used to  
define the record's type \*

Used by: DMENU, PREC1 through PREC6.  
Changed by: DMENU, PREC1 through PREC6.  
Characteristics: I

REQFIL = \* Array containing the name of a file to which  
user requests are written \*

Used by: SVREQ.  
Changed by: SIOPT.  
Characteristics: 6A (3I)

RESC = \* Contains the right-byte value for an ASCII  
ESCAPE character \*

Used by: SESC.  
Changed by: SESC.  
Characteristics: I

RESP = \* Filter impulse response array \*

Used by: PD003  
Changed by: PD005  
Characteristics: (LFILT/2+1)\*F

"RF(NNNN)"= \* Request file containing user request \*

Used by: AOPT  
Changed by: SVREQ  
Characteristics: Sequential disk file containing  
header information followed by user request.

Each request file contains a set of data which is used as if it had been entered from the keyboard in the same order it occurs in the file. It would be nearly impossible to predict all the forms of data which might occur in a request file. But, since all requests are edited before being stored in a request file, the data is seldom likely to be in error (requiring interpretation).

RFROM = \* Contains the right-byte form of a character to be operated on during substitution (from-to) \*

Used by: SCHAR.  
Changed by: SCHAR.  
Characteristics: I

RMASK = \* Mask for isolating the right-byte character in a word \*

Used by: SCHAR.  
Changed by: SCHAR.  
Characteristics: I

ROW = \* Row on terminal screen to which cursor is to be positioned \*

Used by: CURSR.  
Changed by: (modules calling CURSR)  
Characteristics: I

ROWMAX = \* Maximum row value for HP2648A terminal screen \*

Used by: PREC1.  
Changed by: GCOM.  
Characteristics: I

ROWMIN = \* Minimum row value for HP2648A terminal screen \*

Used by: PREC1.

Changed by: GCOM.  
Characteristics: I

RTO = \* Right-byte value of "TO" substitution character (FROM-TO substitution) \*

Used by: SCHAR.  
Changed by: SCHAR.  
Characteristics: I

SARIO = \* Class I/O number assigned to the SAREQ module, used by other modules to uniquely identify SAREQ as the recipient of Class I/O messages \*

Used by: All AD(NNN), PD(NNN), and DD(NNN) programs.  
Changed by: SAREQ.  
Characteristics: I

SCRDAT = \* Screen data buffer array. Used for accepting blocks of data from the screen during block read operations (EXEC calls) \*

Used by: EM001, EM002, EM003, EM004, PARAM.  
Changed by: AOPT, CLRSD.  
Characteristics: 1000A (500I)

SFX = \* Scaling factor to be applied to X signal \*

Used by: PD002.  
Changed by: EM\_\_\_.  
Characteristics: F

SFX is normally set to 1.0, unless scaling of input data is desired.

SFY = \* Scaling factor to be applied to Y signal \*

Used by: PD002.  
Changed by: EM\_\_\_.  
Characteristics: F

SFY is normally set to 1.0, unless scaling of input data is desired.

"SI(NNNN)"= \* Sampled input data file containing raw sampled data \*

Used by: PD001, PD002, PD003, PD004, PD008  
Changed by: AD001, AD002  
Characteristics: Sequential disk file containing header information followed by sample values.

Record (in order):                      Format:



|                       |     |
|-----------------------|-----|
| Identifying text      | 80A |
| Number of data points | I   |
| Sampling frequency    | I   |
| Sample value          | I   |
| (more sample values)  |     |

SS = \* Starting sample number (for offsets within a data block) \*

Used by: PD001.  
 Changed by: EM\_\_\_\_.  
 Characteristics: F

SS will typically be an integer, but PD001 expects it to be a floating-point number for the sake of generality.

STRLEN = \* Length of character string to be operated on; given in number of characters \*

Used by: NOCR.  
 Changed by: NOCR.  
 Characteristics: I

TCONT = \* Holds terminal control characters for taking care of simple chores like clearing the screen \*

Used by: AOPT, BORDR, CLRTC, CURSR, DMENU, DSP, ERMSG, PREC2, PREC5.  
 Changed by: AOPT, BORDR, CLRTC, CURSR, DMENU, DSP, ERMSG, PREC2, PREC5.  
 Characteristics: 20A

TOTPTS = \* Total number of sample points to be processed \*

Used by: PD001.  
 Changed by: EM\_\_\_\_.  
 Characteristics: I

At present, must be a multiple of the number of points per block, PTSBLK.

WDADDR = \* Address of word being manipulated (substitutions, etc.) \*

Used by: NOCR.  
 Changed by: NOCR.  
 Characteristics: I

WINCOR = \* Type of window used by CORR program \*

Used by: PD001.  
Changed by: EM\_\_\_\_.  
Characteristics: I

1 = Rectangular  
2 = Hamming

WTX = \* Positive weighting functions for filter frequency bands \*

Used by: PD005.  
Changed by: EM\_\_\_\_.  
Characteristics: 10F

XSIG = \* Primary set of signal values \*

Used by: PD001, PD002.  
Changed by: EM\_\_\_\_, PD(I).  
Characteristics: (TOTPTS)F

YSIG = \* Secondary (typically optional) signal values \*

Used by: PD001, PD002.  
Changed by: EM\_\_\_\_, PD(I).  
Characteristics: (TOTPTS)F

APPENDIX G  
Data Dictionary  
(Program modules)

Modules are listed alphabetically.

-----

Module name: AD000

Aliases: ACQUIRE\_DATA\_FROM\_FILE

Description: AD000 reads files and provides the data they contain to data processing application programs (PD001, PD002, etc.), acting as though the data were being taken from an A/D converter. This allows the PD(NNN) programs to have a standard interface with AD(NNN) programs, whether the data are coming from file or from a real-time device.

Calling modules: ADATA.

Modules called: (none)

Data items input: ACQIO, PROIO, SARIO.

Data items output: SD(NNNN).

-----

Module name: AD001

Aliases: GENERATE\_DATA

Description: Generate sinusoidal data for use by processing and display application programs.

Data items input: ACQIO, PROIO, SARIO.

Data items output (changed): SD(NNNN).

Calling modules: ADATA.

Modules called: (none)

-----

Module name: AD002

Aliases: SAMPLE\_A/D\_CONVERTER

Description: Take data samples from the analog-to-digital

converter for use by processing and display application programs.

Data items input: ACQIO, PROIO, SARIO.

Data items output (changed): SD(NNNN).

Calling modules: ADATA.

Modules called: (none)

-----

Module name: ADATA

Aliases: ACQUIRE\_DATA

Description: Initiates processing by data acquisition application programs (AD000, AD001, etc.)

Data items input: ACQNO.

Data items output (changed): (none)

Calling modules: SAREQ.

Modules called: AD000, AD001, AD002.

-----

Module name: AOPT

Aliases: ACCEPT\_OPTIONS

Description: AOPT accepts the user's terminal input (options). All data is sent from the terminal in block form, with unit separator (US) characters placed between each field automatically by the terminal.

Two reads of the terminal's data are performed when the user specifies that data entry is complete by hitting either the ENTER key or one of the programmable function keys. The first read checks to see if it was one of the function keys that was hit. If so, and if that key has been assigned a function, its associated flag (such as FHELP) is set. Whether or not a function key was hit, a second read is then performed to get all of the data in protected fields (which requires that the cursor be homed first).

Function keys are identified by their default escape sequence values. Key number 1 sends an ESCp, key number 2 sends an ESCq, etc. If the function keys have been changed from their default settings, a reset of the terminal should be performed before running DSP.

Calling modules: GOPT.

Modules called: CLRRB, CLRSB, CLRTC, PARAM, SCHAR, SESC,  
SIOPT.

Data items input: CONWD1, CONWD2, CONWD3, CURMEN, MAXSCR,  
MAXTCO, NOCRLF, PARBUF, RECBUF.

Data items output (changed): FBACK, FDIREC, FERR,  
FEXIT, FHELP, FRDEF, FREDIS, FSDEF, FSIOPT, MDDCB,  
MENUSE, PARNUM, PARBUF, SCRDATA, TCONT.

-----

Module name: BORDR

Aliases: DRAW\_BORDER

Description: BORDR prepares the terminal screen each time a  
menu is to be displayed. First it clears the screen,  
then it draws a border around it and enables both  
alphabetic and graphic display capabilities.

If it's the first ("Title Page") menu that's being  
displayed (CURMEN = 1), some extra lines, boxes and  
text are added to the basic menu border.

Finally, the current menu (CURMEN) value is written  
to the upper left corner of the screen (mostly as an aid  
to debugging).

Data items input: CONWD2, CURMEN, MAXERT, NOCRLF.

Data items output (changed): ERTEXT, FERR, TCONT.

Calling modules: DMENU, HELP.

Modules called: CLRET, CLRTC, CURSR, SESC.

-----

Module name: CCONS

Aliases: CHECK\_CONSISTENCY

Description: This module is responsible for checking to  
ensure that the user's request is self-consistent.  
For example, if a filter design application program is  
requested, it would make little sense to also specify a  
data acquisition application program.

Data items input: ACQNO, DISNO, PRONO.

Data items output (changed): FCONS, FERR.

Calling modules:  FREQ.

Modules called:  CLRET, ERMSG.

-----

Module name:  CLRET

  Aliases:  CLEAR\_ERTEXT

Description:  CLRET fills the array ERTEXT with nulls (binary zeros).

Calling modules:  (nearly all)

Modules called:  ERMSG.

Data items input:  MAXERT.

Data items output:  ERTEXT, FERR.

-----

Module name:  CLRRB

  Aliases:  CLEAR\_RECBUF

Description:  CLRRB fills the array RECBUF with spaces.

Calling modules:  AOPT, DMENU, ERMSG, HELP, PREC1, PREC2, PREC3, PREC4, PREC5, PREC6.

Modules called:  CLRET, ERMSG.

Data items input:  MAXREC.

Data items output:  ERTEXT, FERR, RECBUF.

-----

Module name:  CLRSD

  Aliases:  CLEAR\_SCRDAT

Description:  CLRSD fills the array SCRDAT with nulls (binary zeros).

Calling modules:  AOPT.

Modules called:  CLRET, ERMSG.

Data items input:  (none)

Data items output: ERTEXT, FERR, SCRDAT.

-----

Module name: CLRTC

Aliases: CLEAR\_TCONT

Description: CLRTC fills the array TCONT with nulls  
(binary zeros).

Calling modules: AOPT, BORDR, CURSR, DMENU, DSP, ERMSG,  
PREC2, PREC3, PREC5.

Modules called: CLRET, ERMSG.

Data items input: MAXTCO.

Data items output: ERTEXT, FERR, TCONT.

-----

Module name: CURSR

Aliases: POSITION\_CURSOR

Description: Given a row/column coordinate pair, CURSR  
positions the screens cursor at that row and column.  
The row must be between 0 and 23; the column must be  
between 0 and 79.

Calling modules: BORDR, ERMSG, HELP, PREC1.

Modules called: CLRET, CLRTC, ERMSG, SESC.

Data items input: COLUMN, CONWD2, NOCRLF, ROW.

Data items output: ERTEXT, FERR, TCONT.

-----

Module name: DD000

Aliases: DISPLAY\_TO\_FILE

Description: DD000 acts just like any of the other data  
display application programs, except it stores the  
data it receives to a file. In this way the data  
acquisition and data processing application programs  
never have to worry about whether they are to provide  
their output to a file or to another program -- they  
always provide it to another program.

Calling modules: DDATA.

Modules called: (none)

Data items input: ACQIO, DISIO, SARIO.

Data items output: (none)

-----

Module name: DD001

Aliases: DISPLAY\_TO\_HP2648

Description: Application program for graphically displaying  
data on HP2648A terminal. Uses the HP2648A's  
intelligent AUTO PLOT function.

Data items input: DISIO, PROIO, SARIO.

Data items output (changed): (none)

Calling modules: DDATA.

Modules called: (none)

-----

Module name: DD002

Aliases: DISPLAY\_TO\_HP1310

Description: Application program to graphically display  
data on the HP1310 terminal.

Data items input: DISIO, PROIO, SARIO.

Data items output (changed): (none)

Calling modules: DDATA.

Modules called: (none)

-----

Module name: DDATA

Aliases: DISPLAY\_DATA

Description: Invokes appropriate application program  
("DD(NNN)") for displaying the results of data acqui-  
sition/processing.



Data items input: DISIO, DISNO, DISREQ, PROIO.

Data items output (changed): FERR.

Calling modules: SAREQ.

Modules called: DD000, DD001, ...

-----

Module name: DIREC

Aliases: PROVIDE\_DIRECTORY

Description: Provides the user with a directory of files  
on disk (according to file type: whether sampled data,  
processed data, etc.)

Data items input: DIRDES, DIRTYP.

Data items output (changed): ERTEXT, FERR.

Calling modules: SIOPT.

Modules called: CLRET, ERMSG.

-----

Module name: DMENU

Aliases: DISPLAY\_MENU

Description: DMENU is responsible for displaying user  
prompting menus. It does this by calling a series of  
subroutines (PREC1 through PREC6), each of which  
is responsible for a part of creating each prompt and  
default item in the menu. Menus are defined by groups  
of records stored in MD(NNN) and CV(NNN) files. PREC1  
through PREC6 are repeatedly called until all the menu-  
defining record groups for the current menu have been  
processed.

Calling modules: GOPT.

Modules called: BORDR, CLRET, CLRRB, CLRTC, ERMSG, PREC1,  
PREC2, PREC3, PREC4, PREC5, PREC6, SCHAR, SESC.

Data items input: CONWD2, CURMEN, FERR, FRDEF, MAXREC,  
MAXTCO, MENUSE, NOCRLF, RECBUF.

Data items output (changed): ERTEXT, FERR, FIRST, RECBUF,

TCONT.

-----

Module name: DSP

Aliases: DIGITAL\_SIGNAL\_PROCESSING

Description: DSP is the main driver for entire system. It repeatedly gets user requests from GREQ and passes them on to SAREQ for satisfaction until either a fatal error occurs or the user asks to exit the system.

Calling modules: (none)

Modules called: CLRTC, GREQ, OLDIO, SAREQ, SESC.

Data items input: ACQNO, CONWD2, DISNO, FERR, FEXIT, MAXTCO, NOCRLF, PRONO.

Data items output (changed): ACQNO, CURMEN, DISNO, FERR, FEXIT, PRONO, TCONT.

-----

Module name: EM001

Aliases: EDIT\_MENU\_1

Description: Edits the user-supplied options from menu number 1. Menu number 1 is special, because it is the only menu from which these options may be selected: save request, read request, directory, and execute request. From menu 1 the user selects which part of the DSP system to enter (acquisition, processing, or display) or what to do with requests that have already been entered (e.g. execute).

Data items input: FERR, SCRDAT.

Data items output (changed): ERTEXT, FDIREC, FEXE, FGEDIT, FREREQ, FSVREQ.

Calling modules: EOPT.

Modules called: CLRET, ERMSG, PARAM.

-----

Module name: EM002

Aliases: EDIT\_MENU\_2

Description: Edits the user-supplied options from menu 2.  
Menu 2 allows the user to select from among various  
data acquisition application programs.

Calling modules: EOPT.

Modules called: CLRET, ERMSG, PARAM.

Data items input: SCRDAT.

Data items output: ACQNO, ERTEXT, FERR, FGEDIT.

-----

Module name: EM003

Aliases: EDIT\_MENU\_3

Description: Edits the user-supplied options from menu 3.  
Menu 3 allows the user to select from among various data  
processing application programs.

Calling modules: EOPT.

Modules called: CLRET, ERMSG, PARAM.

Data items input: SCRDAT.

Data items output: ERTEXT, FERR, FGEDIT, PRONO.

-----

Module name: EM004

Aliases: EDIT\_MENU\_4

Description: Edits the user-supplied options from menu 4.  
Menu 4 allows the user to select from among various data  
display application programs.

Calling modules: EOPT.

Modules called: CLRET, ERMSG, PARAM.

Data items input: SCRDAT.

Data items output: DISNO, ERTEXT, FERR, FGEDIT.

-----

Module name: EM005

Aliases: EDIT\_MENU\_5

Description: Edits the user-supplied options from menu 5.  
Menu 5 allows the user to specify the name of a file  
from which data is to be acquired.

Calling modules: EOPT.

Modules called: ERMSG, PARAM.

Data items input: PARBUF.

Data items output: FERR, FGEDIT, PARNUM, ERTEXT.

-----

Module name: EM006

Aliases: EDIT\_MENU\_6

Description: Edits the user-supplied options from menu 6.  
Menu 6 allows the user to specify the characteristics  
of data to be supplied by the data generation program,  
AD001.

Calling modules: EOPT.

Modules called: ERMSG, NUMER, PARAM.

Data items input: PARBUF.

Data items output: FERR, FGEDIT, PARNUM, ERTEXT.

-----

Module name: EM007

Aliases: EDIT\_MENU\_7

Description: Edits the user-supplied options from menu 7.  
Menu 7 allows the user to specify the parameters to be  
supplied to the FFT program PD004.

Calling modules: EOPT.

Modules called: ERMSG, NUMER, PARAM.

Data items input: PARBUF.

Data items output: FERR, FGEDIT, PARNUM, ERTEXT.

-----

Module name: EM008

Aliases: EDIT\_MENU\_8

Description: Edits the user-supplied options from menu 8.  
Menu 8 allows the user to specify the parameters to be  
supplied to the IFFT program PD006.

Calling modules: EOPT.

Modules called: ERMSG, NUMER, PARAM.

Data items input: PARBUF.

Data items output: FERR, FGEDIT, PARNUM, ERTEXT.

-----

Module name: EM009

Aliases: EDIT\_MENU\_9

Description: Edits the user-supplied options from menu 9.  
Menu 9 allows the user to specify the name of a file  
to which data is to be supplied by the data display  
program DD000.

Calling modules: EOPT.

Modules called: ERMSG, PARAM.

Data items input: PARBUF.

Data items output: FERR, FGEDIT, PARNUM, ERTEXT.

-----

Module name: EM0010

Aliases: EDIT\_MENU\_10

Description: Edits the user-supplied options from menu 10.  
Menu 10 allows the user to specify the data display  
parameters to be supplied to the data display program  
DD001 (displays data to the HP2648A).

Calling modules: EOPT.

Modules called: ERMSG, NUMER, PARAM.

Data items input: PARBUF.

Data items output: FERR, FGEDIT, PARNUM, ERTEXT.

-----  
Module name: EOPT

Aliases: EDIT\_OPTIONS

Description: EOPT call subroutines (EM(NNN)) which edit the user's options . There is one such subroutine for each menu.

Calling modules: GREQ.

Modules called: EM001, EM002,...(one for each menu).

Data items input: ACQNO, CURMEN, DISNO, FDIREC, FERR, FEEXE, FGEDIT, FREREQ, FSVREQ, PRONO.

Data items output: FGEDIT.  
-----

Module name: ERMSG

Aliases: ERROR\_MESSAGE

Description: ERMSG accepts error message text and displays it to the terminal screen in a standard fashion. All error messages are displayed on the last four lines of the screen, with no attention paid to formatting; the user must place carriage returns and line feeds where desired in the midst of the text. Unfortunately, using a "/" in a FORMAT statement for encoding text into the ERTEXT buffer won't work, since a WRITE after calling CODE ignores slashes. It's thus necessary to explicitly place CR/LF sequences in the text.

If the FERR flag is set, signifying a fatal error, then an extra message is printed to tell the user that the program is going to terminate.

Calling modules: (nearly all)

Modules called: CLRRB, CLRTC, CURSR, SESC.

Data items input: CONWD2, ERTEXT, FERR, MAXERT, MAXTCO, NOCRLF, RECBUF.

Data items output (changed): ERTEXT, FERR, TCONT.  
-----

Module name: FA000

Aliases: FORMAT\_FOR\_ACQUISITION\_PROGRAM\_0

Description: FA000 formats the request provided to AD000 to guide its actions when reading data from a file and providing it to data processing/display application programs that are executing concurrently.

Calling modules:  FREQ.

Modules called:  CLRET, ERMSG, FPAR.

Data items input:  PARBUF.

Data items output:  ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name:  FA001

Aliases:  FORMAT\_FOR\_ACQUISITION\_PROGRAM\_1

Description: FA001 formats the request provided to AD001 to guide its actions when it is call upon to generate data.

Calling modules:  FREQ.

Modules called:  CLRET, ERMSG, FPAR.

Data items input:  PARBUF.

Data items output:  ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name:  FA002

Aliases:  FORMAT\_REQUEST\_FOR\_ACQUISITION\_PROGRAM\_2

Description: FA002 formats the request which is provided to AD002 to guide its actions when it is called upon to sample the system's A/D converter for data and provide that data to concurrently operating processing/display application programs.

Calling modules:  FREQ.

Modules called:  CLRET, ERMSG, FPAR.

Data items input:  PARBUF.

Data items output:  ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name: FD000

Aliases: FORMAT\_REQUEST\_FOR\_DISPLAY\_PROGRAM\_0

Description: FD000 formats the request which is provided to DD000 to guide its actions when it is called upon to accept the output of data acquisition/processing application programs and store the data to a file.

Calling modules: FREQ.

Modules called: CLRET, ERMSG, FPAR.

Data items input: PARBUF.

Data items output: ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name: FD001

Aliases: FORMAT\_REQUEST\_FOR\_DISPLAY\_PROGRAM\_1

Description: FD001 formats the request which is provided to DD001 to guide its actions when it is called upon to accept data from acquisition/processing application programs and display the data on the HP2648A terminal.

Calling modules: FREQ.

Modules called: CLRET, ERMSG, FPAR.

Data items input: PARBUF.

Data items output: ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name: FD002

Aliases: FORMAT\_REQUEST\_FOR\_DISPLAY\_PROGRAM\_2

Description: FD002 formats the request which is provided to DD002 to guide its actions when it is called upon to accept data from acquisition/processing application programs and display the data on the HP1300 terminal.

Calling modules: FREQ.

Modules called: CLRET, ERMSG, FPAR.

Data items input: PARBUF.



Data items output: ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name: FPAR

Aliases: FILE\_PARAMETERS

Description: FPAR reads menu data files (either MD(NNN) or CV(NNN) types) for their parameters, and passes back the PARNUMth non-blank parameter (i.e., non zero-length in associated type-4 record) in the PARBUF buffer. FPAR assumes that the file it is to read from has already been opened using the common area file control block MDDCB.

Calling modules: FA000 through FA(NNN), FD000 through FD(NNN), FP001 through FP(NNN).

Modules called: SCOPY.

Data items input: MDDCB, PARBUF, PARNUM.

Data items output: PARBUF.

-----

Module name: FP004

Aliases: FORMAT\_REQUEST\_FOR\_PROCESSING\_PROGRAM\_4

Description: FP004 formats the request which is provided to PD004 to guide its actions when it is called upon to perform an FFT.

Calling modules: FREQ.

Modules called: CLRET, ERMSG, FPAR.

Data items input: PARBUF.

Data items output: ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name: FP006

Aliases: FORMAT\_REQUEST\_FOR\_PROCESSING\_PROGRAM\_6

Description: FP006 formats the request which is provided to PD006 to guide its actions when it is called upon to perform an IFFT.

Calling modules:  FREQ.

Modules called:  CLRET, ERMSG, FPAR.

Data items input:  PARBUF.

Data items output:  ERTEXT, FERR, MDDCB, PARBUF.

-----

Module name:  FREQ

Aliases:  FORMAT\_REQUEST

Description:  Manages modules which format the user's requests so that they are compatible with what the application programs expect.  The raw data FREQ uses is taken from CV(NNN) (current value) files.  The formatted requests are placed in "AREQ", "PREQ(NN)", and "DREQ" files where they can be read by their appropriate application programs.

Each application program has a formatting module associated with it.  Formatting modules FA001 through FA(NNN) are associated with data acquisition application programs AD001 through AD(NNN); FD001 through FD(NNN) are associated with DD001 through DD(NNN); and FP001 through FP(NNN) are associated with PD001 through PD(NNN).

Calling modules:  GREQ.

Modules called:  CLRET, ERMSG, FA000, FA001, ... (one for each acquisition module, with FA000 for acquisition to file); FCONS; FD000, FD001, ... (one for each display module, with FD000 for processing output to file); FP001, FP002,...(one for each processing module).

Data items input:  ACQNO, DISNO, PRONO.

Data items output (changed):  ERTEXT, FERR.

-----

Module name:  GCOM

Aliases:  GET\_REQUEST\_COMMON

Description:  GCOM is only present because of a quirk in HP FORTRAN IV.  In HP FORTRAN, each named common block must be described in a common block subprogram such as GCOM.  There is one benefit to having to do this:  the common data items may be initialized by the subprogram at run time, the subprogram is independently compilable, so

changes can be made to the initial values of common area items by changing and re-compiling this single module.

Calling modules: (none -- system executes automatically)

Modules called: (none)

Data items input: (none)

Data items output: CONWD1, CONWD2, CONWD3, COLMAX, COLMIN, MAXSCR, MAXDCB, MAXMEN, MAXERT, MAXREC, MAXTCO, NOCRLF, ROWMAX, ROWMIN (these are the data items that are initialized at run time).

-----

Module name: GOPT

Aliases: GET\_OPTIONS

Description: GOPT manages the modules that prompt the user for options (via menus) and then accept the user's options. DMENU is called by GOPT to prompt the user, then AOPT is called by GOPT to accept the options.

Calling modules: GREQ.

Modules called: AOPT, DMENU.

Data items input: CURMEN, FERR, FEXIT, FRDEF, FREDIS.

Data items output: FREDIS.

-----

Module name: GREQ

Aliases: GET\_REQUEST

Description: GREQ is responsible for coordinating the activities of modules which get requests from the user. First a set of options is acquired by GOPT, then the options are edited by EOPT, and finally the options are formatted into a request by FREQ.

Calling modules: DSP.

Modules called: EOPT, FREQ, GOPT, TMENT.

Data items input: ACQNO, CURMEN, DISNO, FBACK, FCONS, FDIREC, FERR, FESE, FEXIT, FGEDIT, FREREQ, FSVREQ, PRONO.

Data items output (changed): FBACK.

-----

Module name: HELP

Aliases: (none)

Description: Provides help messages to the user when requested by pressing the HELP function key on the terminal's keyboard.

Data items input: CONWD1, CONWD2, CURMEN, FERR, MAXREC, NOCRLF, RECBUF.

Data items output (changed): ERTEXT, FERR.

Calling modules: SIOPT.

Modules called: BORDR, CLRET, CLRRB, CURSR, ERMSG.

-----

Module name: NOCR

Aliases: NO\_CARRIAGE\_RETURN

Description: NOCR places an underscore character in the user's message string (MESG) just after the character position designated by STRLEN. The 000137B value is placed in the first word after the word containing the character pointed to by STRLEN. Then the value of STRLEN is increased by either two or three to represent the character length of the new string (STRLEN points to the 137B).

It is very important to note that STRLEN is the number of characters in the string, not words.

000137B needs to be placed at the end of text strings because it suppresses the printing of a CR/LF sequence at the end of the line if the string is printed using an EXEC call to DVR05 (the 000137B must be the last word printed for this to work).

Calling modules: PREC5, PREC6.

Modules called: CLRET, ERMSG.

Data items input: MESG, NOCRLF, STRLEN.

Data items output (changed): ERTEXT, FERR, MESG, STRLEN.

-----

Module name: NUMER

Aliases: NUMERIC\_CHECK

Description: NUMER is a logical function subprogram. It returns the value TRUE if PARBUF contains a numeric value in its first NUMCHR character positions. Otherwise, it returns FALSE.

Calling modules: EM006, EM007, EM008, EM010.

Modules called: (none)

Data items input: NUMCHR.

Data items output: PARBUF.

-----

Module name: PARAM

Aliases: EXTRACT\_PARAMETER

Description: PARAM pulls out the requested parameter string from the SCRDAT array, which contains the data read in from the terminal after the user hits the ENTER key or one of the function keys. The input variable PARNUM contains an integer value specifying which of the parameters in SCRDAT, in order, is to be extracted. Parameters are separated by the US character -- 000137B. Once extracted, the parameter is passed back to the calling routine through the PARBUF output variable (without any US characters).

Calling modules: AOPT, TMENT.

Modules called: SCHAR, SCOPY.

Data items input: PARBUF, PARNUM, MAXSCR, SCRDAT.

Data items output: PARBUF.

-----

Module name: PD001

Aliases: AUTO\_CORRELATION, CROSS\_CORRELATION,  
COVARIANCE, POWER\_SPECTRAL\_DENSITY

Description: Application program for performing auto-correlation, auto-covariance, cross-correlation, cross-covariance, and power spectral density functions. See (Ref 10:2.2) and Section 3.4 of this thesis for a

detailed description of the program's design and capabilities.

Data items input: ACQIO, DISIO, PROIO, PROREQ, TOTPTS, PTSBLK, CORTYP, FS, WINCOR, CORPTS, FFTSIZ, XSIG, YSIG, SS.

Data items output: CORVAL, PSDVAL.

Calling modules: PDATA.

Modules called: (none)

-----

Module name: PD002

Aliases: COHERENCE

Description: Application program for performing the coherence function. Two jointly stationary signals (XSIG and YSIG) are processed to derive an estimate of magnitude squared coherence (MSC). See (Ref 10:2.3) and Section 3.4 of this thesis for a detailed description of the program's design and capabilities.

Data items input: ACQIO, DISIO, PROIO, FS, NUMBLK, PTSBLK, SFX, SPY, XSIG, YSIG.

Data items output (changed): CFREQ, MSC, PRONO.

Calling modules: PDATA.

Modules called: (none)

-----

Module name: PD003

Aliases: CONVOLUTION

Description: Application program for performing a convolution. Technique used is overlap-add. See (Ref 10:3.1) and Section 3.4 of this thesis for detailed information on the program's design and capabilities.

Data items input: ACQIO, DISIO, PROIO, IMPFIL, LFILT, RESP, TOTPTS, NUMBLK, PTSBLK.

Data items output (changed): CONDAT.

Calling modules: PDATA.

Modules called: (none)

-----

Module name: PD004

Aliases: FAST\_FOURIER\_TRANSFORM

Description: Application program for performing fast Fourier transform function. See Section 3.4 of this thesis and (Ref 10:1.2) for a detailed description of the program's design and capabilities.

Data items input: ACQIO, DISIO, PROIO, PROREQ, XSIG, FFTSIZ.

Data items output (changed): XSIG.

Calling modules: PDATA.

Modules called: (none)

-----

Module name: PD005

Aliases: FIR\_FILTER\_DESIGN

Description: Application program for designing finite impulse response filters. The Remez exchange algorithm is used to design linear phase FIR filters with minimum weighted Chebyshev error. See Section 3.4 of this thesis and (Ref 10:5.1) for detailed information on the program's design and capabilities.

Data items input: ACQIO, DISIO, PROIO, FFTYPE, IMPFIL, LFILT, EDGE, FX, WTX, LGRID.

Data items output (changed): RESP.

Calling modules: PDATA.

Modules called: (none)

-----

Module name: PD006

Aliases: INVERSE\_FFT

Description: Application program for performing inverse fast Fourier transform. Input is complex, output is real. See Section 3.4 of this thesis and (Ref 10:1.2) for a detailed description of the program's design and

characteristics.

Data items input: ACQIO, DISIO, PROIO, CXSIG, FFTSIZ.

Data items output (changed): XSIG

Calling modules: PDATA

Modules called: none

-----

Module name: PD007

Aliases: IIR\_FILTER\_DESIGN

Description: Application program for designing IIR filters.

Data items input: ACQIO, DISIO, PROIO, PROREQ.

Data items output (changed): PT(NNNN).

Calling modules: PDATA.

Modules called: (none)

-----

Module name: PD008

Aliases: WAVEFORM\_AVERAGE

Description: Application program to perform waveform averaging.

Data items input: ACQIO, DISIO, PROIO, PROREQ.

Data items output (changed): PD(NNNN)

Calling modules: PDATA.

Modules called: (none)

-----

Module name: PDATA

Aliases: PROCESS\_DATA

Description: Initiates data processing application programs (those with names like "PD(NNN)").

Data items input: ACQIO, DISIO, PROIO, PRONO, SARIO.



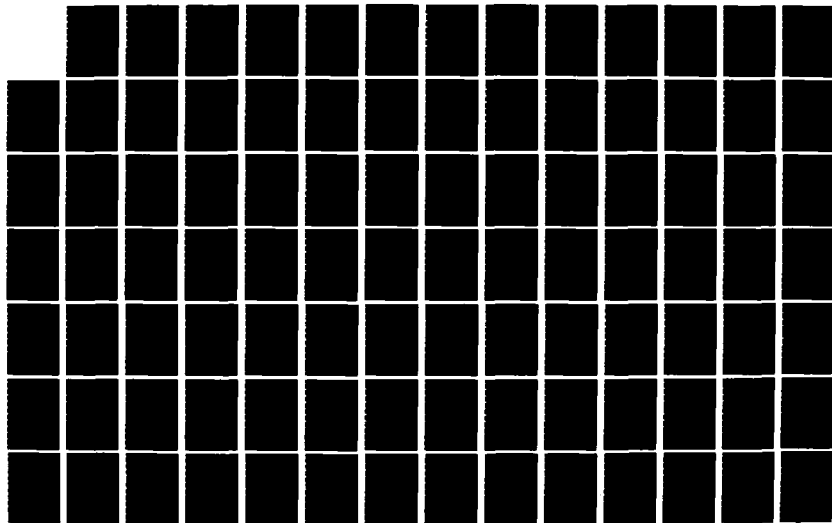
AD-A138 232

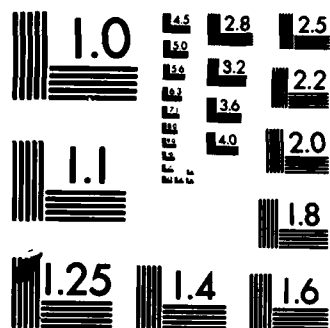
DEVELOPMENT OF A REAL-TIME GENERAL-PURPOSE DIGITAL  
SIGNAL PROCESSING LABO. (U) AIR FORCE INST OF TECH  
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI. J W BENGTON  
DEC 83 AFIT/GCS/EE/83D-3 F/G 9/2

3/4

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

Data items output (changed): ACQIO, DISIO, PROIO.

Calling modules: SAREQ.

Modules called: PD001, PD002, ...

-----

Module name: PREC1

Aliases: PROCESS\_RECORD\_1

Description: PREC1 positions the terminal's cursor in preparation for the writing out of a prompt string as part of a user menu creation. The cursor's position is specified by type-1 records in menu (MD(NNN)) files.

Calling modules: DMENU.

Modules called: CLRET, CLRRB, CURSR, ERMSG.

Data items input: COLMAX, COLMIN, CURMEN, FIRST, MAXREC, MDDCB, RECBUF, ROWMAX, ROWMIN.

Data items output (changed): ERTEXT, FEOF, FERR.

-----

Module name: PREC2

Aliases: PROCESS\_RECORD\_2

Description: PREC2 prepares the user's terminal to print a menu prompting message in either normal or inverse video, according to the content of type-2 records in menu (MD(NNN)) files. If the type-2 record specifies inverse video for a prompt message, the proper escape sequence is sent to the terminal; otherwise, nothing is done.

Calling modules: DMENU.

Modules called: CLRET, CLRRB, CLRTC, ERMSG, SESC.

Data items input: CONWD2, CURMEN, MAXREC, MAXTCO, MDDCB, NOCRLF, RECBUF, RECTYP, TCONT.

Data items output (changed): ERTEXT, FERR, TCONT.

-----

Module name: PREC3

Aliases: PROCESS\_RECORD\_3

Description: PREC3 displays the prompting text given by type-3 records in menu (MD(NNN)) files. Inverse video is also turned off at the end of the text, just in case it was turned on by PREC2.

Data items input: CONWD2, CURMEN, MAXREC, MAXTCO, MDDCB, NOCRLF, RECBUF, TCONT.

Data items output (changed): ERTEXT, FERR, TCONT.

Calling modules: DMENU.

Modules called: CLRET, CLRRB, CLRTC, ERMSG, SESC.

-----

Module name: PREC4

Aliases: PROCESS\_RECORD\_4

Description: Sets up unprotected fields (the only ones data may be entered into) on a menu display. The field length is provided by a type-4 record in a menu (MD(NNN)) file.

Calling modules: DMENU.

Modules called: CLRET, CLRRB, ERMSG.

Data items input: CURMEN, MAXREC, MDDCB, RECBUF.

Data items output (changed): ERTEXT, FERR, FLDLEN.

-----

Module name: PREC5

Aliases: PROCESS\_RECORD\_5

Description: PREC5 turns on the terminal's inverse video function (for the default value to be printed by PREC6) if it has been requested by a type-5 record in a menu (MD(NNN)) files.

Calling modules: DMENU.

Modules called: CLRET, CLRRB, CLRTC, ERMSG, NOCR, SESC.

Data items input: CONWD2, CURMEN, LENCHR, MAXREC, MAXTCO, MDDCB, NOCRLF, RECBUF.

Data items output (changed): ERTEXT, FERR, TCONT.

-----

Module name: PREC6

Aliases: PROCESS\_RECORD\_6

Description: PREC6 takes the default values provided by type-6 records (in MD(NNN) files) and writes them to the protected fields set up by PREC3. The default values are truncated, if need be, to fit into the unprotected default fields.

Calling modules: DMENU

Modules called: CLRRB, NOCR, SCHAR.

Data items input: CONWD2, MAXREC, MDDCB, RECBUF.

Data items output (changed): FERR.

-----

Module name: SAREQ

Aliases: SATISFY\_REQUEST

Description: SAREQ is responsible for managing the satisfaction of user requests by calling modules (ADATA, PDATA, DDATA) which execute application programs for data acquisition, processing, and display.

Calling modules: DSP.

Modules called: ADATA, PDATA, DDATA.

Data items input: ACQNO, DISNO, FERR, PRONO.

Data items output (changed): ACQIO, ACQNO, DISIO, DISNO, PROIO, PRONO, SARIO.

-----

Module name: SCHAR

Aliases: SUBSTITUTE\_CHARACTER

Description: SCHAR replaces all occurrences of the character LFROM with the character LTO in the string MSG, starting at character position CBEGIN and ending at character position CEND. (Note that CBEGIN and CEND point to

characters, not words).

LFROM and LTO are the left-byte values of their respective characters. E.g., the character "X" is an octal 05400B in the left byte of a word (see the character set table in the back of the HP FORTRAN IV manual for a complete list of left and right-byte ASCII character codes). LFROM and LTO should be assigned their values in the calling routine using statements like:

```
LFROM = LHX (or) LFROM = 054000B
LTO = LHY LTO = 054400B
```

These assignments will change X's in the string to Y's. The right byte is ignored; it is masked out before any other operations are performed.

The string is currently limited to being 80 characters or less in length.

Calling modules: AOPT, DMENU, PREC6, TMENT.

Modules called: CLRET, ERMSG.

Data items input: CBEGIN, CEND, LFROM, LTO, MESH.

Data items output (changed): ERTEXT, MESH.

-----

Module name: SCOPY

Aliases: STRING\_COPY

Description: SCOPY copies a string from its input buffer to its output buffer. It is used mostly to copy parameters from the array SCRDATA (which contains terminal input) into the array RECBUF, where the parameters can be dealt with one at a time for editing or formatting purposes. SCOPY allows does not require the input string to begin on a word boundary; nor does it require that the receiving buffer accept the string starting on a word boundary (left and right-byte character values are automatically transposed as necessary). IBEGIN specifies the first character of the input string; IEND specifies the last character. OBEGIN specifies the beginning character position for storing the string in the output buffer (the string length is computed as IEND-IBEGIN+1).

Calling modules: PARAM.

Modules called: CLRET, ERTEXT.

Data items input: IBEGIN, IEND, INBUF, OBEGIN.

Data items output: FERR, OBUF.

-----

Module name: SESC

Aliases: SUBSTITUTE\_ESCAPE

Description: SESC takes a string composed of printing ASCII characters and replaces all occurrences of capital "E" with an ASCII escape character. This allows terminal control sequences to be formed by encoding FORMAT character strings, instead of having to deal with directly encoding the octal value for an escape into the string.

Calling modules: AOPT, BORDR, CURSR, DMENU, DSP, PREC2, PREC5.

Modules called: CLRET, ERMSG.

Data items input: LENWDS, MMSG.

Data items output: ERTEXT, FERR, MMSG.

-----

Module name: SIOPT

Aliases: SATISFY\_IMMEDIATE\_OPTIONS

Description: Manages modules responsible for satisfying immediate user options. Immediate user options are those which may be satisfied without having to perform any data acquisition, processing, or display (e.g. help, saving new defaults, etc.)

Calling modules: AOPT.

Modules called: DIREC, HELP, SVREQ, TMENT.

Data items input: CURMEN, FBACK, FDIREC, FERR, FEXIT, FHELP, FRDEF, FSDEF, FSVREQ.

Data items output (changed): DIRDES, DIRTYP, FERR, REQFIL.

-----

Module name: SVREQ

Aliases: SAVE\_REQUEST

Description: Save the user's request for later use. "User's request" means all current values for menus which have been accessed during the terminal session.

Data items input: REQFIL.

Data items output (changed): ERTEXT, FERR.

Calling modules: SIOPT.

Modules called: CLRET, ERMSG.

-----

Module name: TMENT

Aliases: TRAVERSE\_MENU\_TREE

Description: Coordinates the traversal of the menu tree. Arranges for proper menus to be displayed according to the user's directions.

TMENT recognizes two types of menus: decision menus and data entry menus. Decision menus are expected to request that the user enter an integer to specify a choice of options. For each possible option, there should be an entry in the MENUT file to relate the option to its own menu. If the user chooses option N while viewing menu M, there should be an Nth item in the MENUT file of the form "M,K" where K is the menu to be displayed for data entry (or further decisions related to option N. ("M,K" should be the Nth item to have "M" as its first integer -- not the Nth item in the file overall). Data entry menus are expected to have at most one subordinate menu, and thus will have only one entry in the MENUT file.

Data items input: CURMEN, FBACK, PARBUF.

Data items output (changed): CURMEN, ERTEXT, FERR.

Calling modules: SIOPT, GREQ.

Modules called: CLRET, ERMSG, PARAM, SCHAR.



APPENDIX H  
User Interface Software Listings

PROGRAM DSP(),20 OCT 83, V1.2

```
C*****
C*
C* Module name: DSP (DIGITAL_SIGNAL_PROCESSING)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 20 Oct 83
C*
C* Description:
C*
C* DSP is the main driver for the entire system.
C* It repeatedly gets user requests from GREQ and
C* passes them on to SAREQ for satisfaction until
C* either a fatal error occurs or the user asks to
C* get out of the system.
C*
C* Calling module: (none)
C*
C* Modules called: CLRTC,GREQ,OLDIO,SAREQ,SESC.
C*
C* Data items input: ACQNO, CONWD2, DISNO, FERR, FEXIT,
C* MAXTCO, NOCRLF, PRONO.
C*
C* Data items output (changed): CURMEN, FERR, FEXIT, LEN,
C* TCONT, ACQNO, DISNO, PRONO.
C*
C*****
```

C Common block definitions.

```
COMMON /COM2/MDDCB
COMMON /COM6/TCONT
COMMON /COM22/CONWD2
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF

INTEGER MDDCB(144),TCONT(10)
INTEGER CONWD2,MAXTCO,NOCRLF
```

C Normal data item definitions.

```
INTEGER PRONO(10)
INTEGER ACQNO,CURMEN,DISNO,RESET
LOGICAL FERR,FEXIT
```

C Initializations.

```

CALL OLDIO
FERR = .FALSE.
FEXIT = .FALSE.
CURMEN = 1
ACQNO = 1
DO 5 I = 1,10
 PRONO(I) = 0
5 CONTINUE
 PRONO(1) = 4
 PRONO(2) = 6
 DISNO = 1

C Main loop. Executed until fatal error or user request to
C exit. First get a request.

10 CALL GREQ(ACQNO,CURMEN,DISNO,FERR,FEXIT,PRONO)

 IF ((.NOT.FERR).AND.(.NOT.FEXIT)) GOTO 20
 GOTO 10101

C If no fatal error and no request to exit, satisfy request.

20 CALL SAREQ(ACQNO,DISNO,FERR,PRONO)

C If no fatal error encountered in satisfying request, get
C another request.

 IF (.NOT.FERR) GOTO 10

C Turn off block and format modes, clear screen,
C turn off graphics display, and close files.

10101 CALL CLRTC
 CALL CODE
 WRITE(TCONT,9999)
 LEN = MAXTCO
 CALL SESC(FERR,TCONT,LEN)
 TCONT(LEN) = NOCRLF
 CALL EXEC(2,CONWD2,TCONT,LEN)
 CALL CLOSE(MDDCB)
 WRITE(1,9998)

 STOP

9999 FORMAT("EXE&k0BEHEJE*ddz")
9998 FORMAT(20X,"(DSP) ...dust to dust...")
9997 FORMAT(A2)

 END
 END$

```

```

SUBROUTINE GREQ(ACQNO,CURMEN,DISNO,FERR,FEXIT,PRONO)
-,15 OCT 83 V1.2

```

```

C*****
C*
C* Module name: GREQ (GET_REQUEST)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 15 Oct 83
C*
C* Description:
C*
C* GREQ is responsible for coordinating the activities
C* of modules which get requests from the user. First
C* a set of options is acquired (GOPT), then the options
C* are edited (EOPT), and finally the options are
C* formatted into requests (FREQ). TMENT is called in
C* case the entry of options for a request requires
C* more than one menu.
C*
C* Calling module: DSP.
C*
C* Modules called: EOPT, FREQ, GOPT, TMENT.
C*
C* Data items input: ACQNO,CURMEN,DISNO,FBACK,FCONS,
C* FDIREC, FERR, FESE, FEXIT, FGEDIT, FREREQ, FSVREQ,
C* PRONO.
C*
C* Data items output (changed): ACQNO, DISNO, FBACK,
C* PRONO.
C*
C*****

```

C Normal data item definitions.

```

INTEGER ACQNO,CURMEN,DISNO,PRONO(10)
LOGICAL FCONS,FERR,FESE,FEXIT,FGEDIT,FBACK
LOGICAL FDIREC,FESE,FREREQ,FSVREQ

```

C Main loop; repeat until fatal error or user desires  
C to exit.

C Start off by getting a menu full of options.

```

20 CALL GOPT(CURMEN,FERR,FEXIT)
IF (FERR.OR.FEXIT) GOTO 10101

```

C Edit the user's options

```

CALL EOPT(ACQNO,CURMEN,DISNO,FDIREC,FERR,FESE,
- FGEDIT,FREREQ,FSVREQ,PRONO)
IF (FERR) GOTO 10101

```

C If the options didn't pass editing criteria, go back  
C and try again by asking for option reentry.

IF (FERR) GOTO 10101  
IF (.NOT.FGEDIT) GOTO 20

C If the user hasn't asked to execute the options entered  
C thus far, keep accepting options.

IF (FEXE) GOTO 30  
FBACK = .FALSE.  
CALL TMENT(CURMEN,FBACK,FERR)  
GOTO 20

C Format the user's options into requests and return to  
C DSP so that the requests may be satisfied.

30 CALL FREQ(ACQNO,DISNO,FCONS,PRONO,FERR)  
FEXE = .FALSE.

10101 RETURN

END  
END\$

SUBROUTINE GOPT(CURMEN,FERR,FEXIT),27 OCT 83 V1.3

```

C*****
C*
C* Module name: GOPT (GET_OPTIONS)
C* Author: Capt John Bengtson
C* Version: 1.3
C* Date: 27 Oct 83
C*
C* Description:
C*
C* GOPT manages the modules that prompt the user (via
C* menus) and then accept the user's options (response
C* data). DMENU does the prompting, and AOPT does the
C* accepting.
C*
C* Calling module: GREQ.
C*
C* Modules called: AOPT, DMENU.
C*
C* Data items input: CURMEN, FERR, FEXIT, FRDEF, FREDIS.
C*
C* Data items output (changed): FREDIS.
C*
C*****

```

C Normal data item definitions.

```

 INTEGER CURMEN
 LOGICAL FERR,FEXIT,FRDEF,FREDIS

```

C Main program section. Start by displaying a menu.

```

10 CALL DMENU(CURMEN,FERR,FRDEF)
 IF (FERR) GOTO 10101

```

C If no fatal error was encountered in displaying the menu,  
C accept the user's options.

```

 FREDIS = .FALSE.
 CALL AOPT(CURMEN,FERR,FEXIT,FRDEF,FREDIS)

```

C If the user made an immediate request, re-display the  
C current menu.

```

 IF ((FREDIS).AND.(.NOT.FERR)) GOTO 10

```

```

10101 RETURN

```

```

 END
 END$

```

SUBROUTINE DMENU(CURMEN,FERR,FRDEF)  
 -,29 OCT 83 V1.2

```

C*****
C*
C* Module name: DMENU (DISPLAY MENU)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 29 Oct 83
C*
C* Description:
C*
C* DMENU is responsible for displaying prompt
C* menus. It does this by calling a series of sub-
C* routines (PREC1 -- PREC6), each of which sends
C* a certain type of data item to the terminal
C* screen. The subroutines are called an arbitrary
C* number of times, depending upon the number of
C* items in the menu.
C*
C* Calling module: GOPT.
C*
C* Modules called: BORDR, CLRET, CLRRB, CLRTC, ERMSG,
C* PREC1 through PREC6, SCHAR, SESC.
C*
C* Data items input: CONWD2, CURMEN, FERR, FRDEF, MAXREC,
C* MAXTCO, MENUSE, NOCRLF, RECBUF.
C*
C* Data items output (changed): ERTEXT, FERR, FIRST,
C* RECBUF, TCONT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM3/MENUSE
COMMON /COM4/RECBUF
COMMON /COM6/TCONT
COMMON /COM22/CONWD2
COMMON /COM30/MAXREC
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
LOGICAL MENUSE(100)
INTEGER RECBUF(40)
INTEGER TCONT(10)

INTEGER CONWD2

```

```
INTEGER MAXREC
INTEGER MAXTCO
INTEGER NOCRLF
```

C Normal data item definitions.

```
INTEGER MDNAM(3)
INTEGER ATTRI,CURMEN,DISCOL,DISROW,FLDLN,RECTYP
LOGICAL FEOF,FERR,FIRST,FRDEF
```

C Initializations.

C When ANDed with a word, LMASK will leave only the left  
C character of that word intact (masking out the right  
C character).

```
DATA LMASK/077400B/
```

C Main program section.

C Prepare the screen for receiving a menu.

```
CALL BORDR(CURMEN,FERR)
```

C Encode proper file name for use by OPEN statement.

```
IF ((.NOT.MENUSE(CURMEN)).OR.FRDEF) GOTO 10
CALL CODE
WRITE(MDNAM,9998) CURMEN
GOTO 20
```

```
10 CALL CODE
WRITE(MDNAM,9999) CURMEN
```

C Replace blanks in file name with zeros.

```
20 CALL SCHAR(FERR,1H ,1H0,1,5,MDNAM)
```

C Open the menu file to be read (either MD or CV type).

```
CALL OPEN(MDDCB,IERR,MDNAM)
IF (IERR.GE.0) GOTO 30
FERR = .TRUE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9997) (MDNAM(I),I=1,3)
CALL ERMSG(FERR)
GOTO 10101
```

C Read until first non-type-0 record is found or EOF is reac

```
30 CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
CALL CODE
```

```

 READ(RECBUF,*) RECTYP
 IF ((RECTYP.EQ.0).AND.(LEN.GE.0)) GOTO 30

C Take care of file which is empty or contains nothing
C but type-0 records.

 IF (LEN.GE.0) GOTO 40
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9995) CURMEN
 CALL ERMSG(FERR)
 GOTO 10101

C Take care of file which doesn't have type 1 record as
C first non type-0 record.

40 IF (RECTYP.EQ.1) GOTO 50
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9991) CURMEN,(RECBUF(I),I=1,MAXREC)
 CALL ERMSG(FERR)
 GOTO 10101

C MAIN LOOP: start processing records.

50 FIRST = .TRUE.
 FEOF = .FALSE.

C Write everything but default values to screen (all unpro-
C tected fields must be defined before data can be written
C into them).

60 CALL PREC1(CURMEN,FEOF,FERR,FIRST)
 IF (FERR) GOTO 10101
 IF (FEOF) GOTO 80
 CALL PREC2(CURMEN,FERR)
 IF (FERR) GOTO 10101
 CALL PREC3(CURMEN,FERR)
 IF (FERR) GOTO 10101
 CALL PREC4(CURMEN,FERR,FLDLN)
 IF (FERR) GOTO 10101
 CALL PREC5(CURMEN,FERR,FLDLN)
 IF (FERR) GOTO 10101

C Read over type-6 record.

 CALL CLRRB
 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
 IF (LEN.GE.0) GOTO 60
 FERR = .TRUE.

```



```

 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9992) CURMEN,(RECBUF(I),I=1,MAXREC)
 CALL ERMSG(FERR)
 GOTO 10101

C Set terminal to block and format mode and home cursor in
C preparation for writing out default values.

80 CALL CLRTC
 CALL CODE
 WRITE(TCONT,9994)
 LEN = MAXTCO
 TCONT(LEN) = NOCRLF
 CALL SESC(FERR,TCONT,LEN)
 CALL EXEC(2,CONWD2,TCONT,LEN)

C Rewind data file so that type 6 records (which contain
C default values) may be read, and read to first type-3
C record.

 CALL RWNDF(MDDCB)
 FEOF = .FALSE.
90 CALL CLRRB
 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
 IF (LEN.GE.0) GOTO 100
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9992) CURMEN,(RECBUF(I),I=1,MAXREC)
 CALL ERMSG(FERR)
 GOTO 10101

100 CALL CODE
 READ(RECBUF,*) RECTYP
 IF (RECTYP.NE.3) GOTO 90

C Now process all type-6 records.

110 CALL PREC6(FEOF,FERR)
 IF ((.NOT.FERR).AND.(.NOT.FEOF)) GOTO 110

10101 CALL CLOSE(MDDCB)
 RETURN

9999 FORMAT("MD",I3," ")
9998 FORMAT("CV",I3," ")
9997 FORMAT(10X,"(DMENU) Needed menu file ",3A2
- ," doesn't exist.")
9995 FORMAT(10X,"(DMENU) No type-1 records found in menu fi
- ,I3)
9994 FORMAT("EHE&k 1BEW")

```

9992 FORMAT(5X,"(DMENU) Menu records out of order in menu f  
- ,I3,"; expected type-6 record, found instead: ",40A  
9991 FORMAT(5X,"(DMENU) Menu records out of order in menu f  
- ,I3,"; expected type-1 record, found instead: ",40A  
  
END  
END\$

SUBROUTINE PREC1(CURMEN, FEOF, FERR, FIRST)  
 -,1 NOV 83, V1.2

```

C*****
C*
C* Module name: PREC1 (PROCESS_RECORD_1)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 1 Nov 83
C*
C* Description:
C*
C* Performs the cursor positioning function which is
C* specified by type-1 records in menu (MD(NNN)) files.
C*
C* Calling modules: DMENU.
C*
C* Modules called: CLRET, CLRRB, CURSR, ERMSG.
C*
C* Data items input: COLMAX, COLMIN, CURMEN, FIRST,
C* MAXREC, MDDCB, RECBUF, ROWMAX, ROWMIN.
C*
C* Data items output (changed): ERTEXT, FEOF, FERR.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM4/RECBUF

```

```

COMMON /COM24/COLMAX
COMMON /COM25/COLMIN
COMMON /COM30/MAXREC
COMMON /COM33/ROWMAX
COMMON /COM34/ROWMIN

```

```

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
INTEGER RECBUF(40)

```

```

INTEGER COLMAX
INTEGER COLMIN
INTEGER MAXREC
INTEGER ROWMAX
INTEGER ROWMIN

```

C Normal data item definitions.

```

INTEGER CURMEN, DISCOL, DISROW, RECTYP

```

LOGICAL FEOF,FERR,FIRST

C Initializations.

C Main program section.

C If FIRST is true, it means that a non-type-0 record has  
C already been read by DMENU and is held in RECBUF.

```
LEN = 1
IF (FIRST) GOTO 10
CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
GOTO 20
```

10 FIRST = .FALSE.

C Exit if EOF has been reached.

```
20 IF (LEN.GE.0) GOTO 25
 FEOF = .TRUE.
 GOTO 10101
```

```
25 CALL CODE
 READ(RECBUF,*) RECTYP,DISROW,DISCOL
```

C Make sure proper record type.

```
IF (RECTYP.EQ.1) GOTO 30
FERR = .TRUE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999) CURMEN,(RECBUF(I),I=1,MAXREC)
CALL ERMSG(FERR)
GOTO 10101
```

```
30 IF ((DISROW.GE.ROWMIN).AND.(DISROW.LE.ROWMAX)) GOTO 40
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998) CURMEN,(RECBUF(I),I=1,MAXREC)
 CALL ERMSG(FERR)
 GOTO 10101
```

```
40 IF ((DISCOL.GE.COLMIN).AND.(DISCOL.LE.COLMAX)) GOTO 50
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9997) CURMEN,(RECBUF(I),I=1,MAXREC)
 CALL ERMSG(FERR)
 GOTO 10101
```

C Position cursor.

```

50 CALL CURSR(DISROW,DISCOL,FERR)

10101 RETURN

9999 FORMAT("(PREC1) Menu records out of order for menu "
- ,I3,"; expected type 1 record, found instead: ",/
- ,40A2)
9998 FORMAT("(PREC1) Row coordinate out of bounds on type",
- " 1 record for menu ",I3,"; record in error: ",/
- ,40A2)
9997 FORMAT("(PREC1) Column coordinate out of bounds on",
- " type 1 record for menu ",I3," type 1 record in",
- " error: ",/,40A2)
9996 FORMAT("E&a ",I2,"r ",I2," C")

 END
 END$

```

SUBROUTINE PREC2(CURMEN,FERR)  
 -,1 NOV 83, V1.2

```

C*****
C*
C* Module name: PREC2 (PROCESS_RECORD_2)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 1 Nov 83
C*
C* Description:
C*
C* Sets up the user's terminal to print a menu
C* prompting message in either normal or inverse video,
C* according to the content of type-2 records in menu
C* (MD(NNN)) files.
C*
C* Calling modules: DMENU.
C*
C* Modules called: CLRET, CLRRB, CLRTC, ERMSG, SESC.
C*
C* Data items input: CONWD2, CURMEN, MAXREC, MAXTCO,
C* MDDCB, NOCRLF, RECBUF, RECTYP, TCONT.
C*
C* Data items output (changed): ERTEXT, FERR, TCONT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM4/RECBUF
COMMON /COM6/TCONT

COMMON /COM22/CONWD2
COMMON /COM30/MAXREC
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
INTEGER RECBUF(40)
INTEGER TCONT(10)

INTEGER CONWD2
INTEGER MAXREC
INTEGER MAXTCO
INTEGER NOCRLF

```

C Normal data item definitions.

INTEGER ATTRI,CURMEN,RECTYP  
LOGICAL FERR

C Initializations.

C Main program section.

C Read a record and decode it.

CALL CLRRB  
CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)  
CALL CODE  
READ(RECBUF,9999) RECTYP,ATTRI

C Make sure it's the proper record type and that EOF has  
C not been reached.

IF ((RECTYP.EQ.2).AND.(LEN.GE.0)) GOTO 10  
FERR = .TRUE.  
CALL CLRET  
CALL CODE  
WRITE(ERTEXT,9998) CURMEN,(RECBUF(I),I=1,MAXREC)  
CALL ERMSG(FERR)  
GOTO 10101

C Only "I" and "N" are acceptable as attribute characters.

10 IF ((ATTRI.EQ.000111B).OR.(ATTRI.EQ.000116B).OR.  
- (ATTRI.EQ.000040B).OR.(ATTRI.EQ.0)) GOTO 20  
FERR = .TRUE.  
CALL CLRET  
CALL CODE  
WRITE(ERTEXT,9997) CURMEN,(RECBUF(I),I=1,MAXREC)  
CALL ERMSG(FERR)  
GOTO 10101

C Write out attribute-defining character sequence.

20 IF (ATTRI.NE.111B) GOTO 10101  
CALL CLRTC  
CALL CODE  
WRITE(TCONT,9996)  
C LEN = MAXTCO  
TCONT(10) = NOCRLF  
CALL SESC(FERR,TCONT,10)  
CALL EXEC(2,CONWD2,TCONT,10)

10101 RETURN

9999 FORMAT(I1,1X,A1)

9998 FORMAT("(PREC2) Menu records out of order for menu ",I

- "; expected type 2 record, found instead: ",40A2)  
9997 FORMAT("(PREC2) Illegal prompt attribute in menu ",I3,  
- "; should be I, N or blank, found instead: ",40A2)  
9996 FORMAT("E&dB")

END

END\$



SUBROUTINE PREC3(CURMEN,FERR)  
-,1 NOV 83, V1.2

```
C*****
C*
C* Module name: PREC3 (PROCESS_RECORD_3)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 1 Nov 83
C*
C* Description:
C*
C* Displays the prompting text given by type-3 records
C* in menu (MD(NNN)) files.
C*
C* Calling modules: DMENU.
C*
C* Modules called: CLRET, CLRRB, CLRTC, ERMSG, SESC.
C*
C* Data items input: CONWD2, CURMEN, MAXREC, MAXTCO,
C* MDDCB, NOCRLF, RECBUF, RECTYP, TCONT.
C*
C* Data items output (changed): ERTEXT, FERR, TCONT.
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM4/RECBUF
COMMON /COM6/TCONT
```

```
COMMON /COM22/CONWD2
COMMON /COM30/MAXREC
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF
```

```
INTEGER ERTEXT(80)
INTEGER MDDCB(144)
INTEGER RECBUF(40)
INTEGER TCONT(10)
```

```
INTEGER CONWD2
INTEGER MAXREC
INTEGER MAXTCO
INTEGER NOCRLF
```

C Normal data item definitions.

```
INTEGER CURMEN,LENWDS
INTEGER RECLen,RECTYP
```

# LOGICAL FERR

C Main program section.

C Read a record and decode it.

```
CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
CALL CODE
READ(RECBUF,9999)RECTYP
```

C Make sure it's the proper record type and that EOF has  
C not been reached.

```
IF ((RECTYP.EQ.3).AND.(LEN.GE.0)) GOTO 10
FERR = .TRUE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9998) CURMEN,(RECBUF(I),I=1,MAXREC)
CALL ERMSG(FERR)
GOTO 10101
```

C Shift RECBUF one word left to erase "3,"

```
10 DO 20 I = 1,MAXREC-1
20 RECBUF(I) = RECBUF(I+1)
```

C Write out prompting text (remember that LEN is number  
C of words read, and text has been shifted so no text  
C is being overwritten by NOCRLF).

```
RECBUF(LEN) = NOCRLF
CALL EXEC(2,CONWD2,RECBUF,LEN)
```

C Also turn off inverse video, which may be on.

```
CALL CLRTC
CALL CODE
WRITE(TCONT,9997)
LENWDS = MAXTCO
CALL SESC(FERR,TCONT,LENWDS)
TCONT(LENWDS) = NOCRLF
CALL EXEC(2,CONWD2,TCONT,LENWDS)
```

10101 RETURN

```
9999 FORMAT(I1,1X,39A2)
9998 FORMAT("(PREC3) Menu records out of order for menu "
- ,I3,"; expected type 3 record, found instead: ",/,
- 40A2)
9997 FORMAT("E&d@")
```

```
END
END$
```



SUBROUTINE PREC4(CURMEN,FERR,FLDLEN)  
-,1 NOV 83 V1.2

```
C*****
C*
C* Module name: PREC4 (PROCESS_RECORD_4)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 1 Nov 83
C*
C* Description:
C*
C* PREC4 sets up unprotected fields (the only ones
C* data may be entered into) on a menu display. The
C* field length is provided by type-4 records in menu
C* (MD(NNN)) files.
C*
C* Calling modules: DMENU.
C*
C* Modules called: CLRET, CLRB, ERMSG.
C*
C* Data items input: CURMEN, MAXREC, MDDCB, RECBUF.
C*
C* Data items output (changed): ERTEXT, FERR, FLDLEN.
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM4/RECBUF

COMMON /COM30/MAXREC

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
INTEGER RECBUF(40)

INTEGER MAXREC
```

C Normal data item definitions.

```
INTEGER CURMEN,RECTYP,FLDLEN
LOGICAL FERR
```

C Main program section.

C Read a record and decode it.

```
CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
CALL CODE
```

```

 READ(RECBUF,9997) RECTYP,FLDLEN

C Make sure it's the proper record type and that EOF
C has not been reached.

 IF ((RECTYP.EQ.4).AND.(LEN.GE.0)) GOTO 10
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) CURMEN,(RECBUF(I),I=1,MAXREC)
 CALL ERMSG(FERR)
 GOTO 10101

C Make sure response field length is between 0 and 78.
10 IF ((FLDLEN.GE.0).AND.(FLDLEN.LE.78)) GOTO 10101
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998) CURMEN,(RECBUF(I),I=1,MAXREC)
 CALL ERMSG(FERR)

10101 RETURN

9999 FORMAT("(PREC4) Menu records out of order for menu "
- ,I3,"; expected type 4 record, found instead:","/,40A2)
9998 FORMAT("(PREC4) Field length must be between 0 and ",
- "78 in menu ",I3,"; found instead:","/,40A2)
9997 FORMAT(I1,1X,I3)

 END
 END$

```

SUBROUTINE PREC5(CURMEN,FERR,LENCHR)  
 -,1 NOV 83, V1.2

```

C*****
C*
C* Module name: PREC5 (PROCESS_RECORD_5)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 1 Nov 83
C*
C* Description:
C*
C* PREC5 sets up the user's terminal to print a menu
C* default value message in either normal or inverse
C* video, according to the content of type-5 records
C* in menu (MD(NNN)) files.
C*
C* Calling modules: DMENU.
C*
C* Modules called: CLRET, CLRRB, CLRTC, ERMSG, NOCR, SESC.
C*
C* Data items input: CONWD2, CURMEN, LENCHR, MAXREC,
C* MAXTCO, MDDCB, NOCRLF, RECBUF.
C*
C* Data items output (changed): ERTEXT, FERR, TCONT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM4/RECBUF
COMMON /COM6/TCONT

```

```

COMMON /COM22/CONWD2
COMMON /COM30/MAXREC
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF

```

```

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
INTEGER RECBUF(40)
INTEGER TCONT(10)

```

```

INTEGER CONWD2
INTEGER MAXREC
INTEGER MAXTCO
INTEGER NOCRLF

```

C Normal data item definitions.

```
INTEGER ATTRI,CURMEN,LENCHR
INTEGER RECTYP,SPACE
LOGICAL FERR
```

```
C Main program section.
C Read a record and decode it.
```

```
CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
CALL CODE
READ(RECBUF,9999) RECTYP,ATTRI
```

```
C Make sure it's the proper record type and that EOF
C has not been reached.
```

```
IF ((RECTYP.EQ.5).AND.(LEN.GE.0)) GOTO 10
FERR = .TRUE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9998) CURMEN,(RECBUF(I),I=1,MAXREC)
CALL ERMSG(FERR)
GOTO 10101
```

```
C Only "I" and "N" are acceptable as attribute characters.
C (Don't even check if the length of the string is zero.)
```

```
10 IF (LENCHR.LE.0) GOTO 10101
IF ((ATTRI.EQ.000111B).OR.(ATTRI.EQ.000116B).OR.
- (ATTRI.EQ.40B).OR.(ATTRI.EQ.0)) GOTO 20
FERR = .TRUE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9997) CURMEN,(RECBUF(I),I=1,MAXREC)
CALL ERMSG(FERR)
GOTO 10101
```

```
C Write out characters to define the beginning of an
C unprotected field.
```

```
20 CALL CLRTC
CALL CODE
WRITE(TCONT,9996)
LEN = MAXTCO
TCONT(LEN) = NOCRLF
CALL SESC(FERR,TCONT,LEN)
CALL EXEC(2,CONWD2,TCONT,LEN)
```

```
C If specified, write out characters to define inverse video
C field.
```

```
IF (ATTRI.NE.111B) GOTO 30
CALL CLRTC
```

```

CALL CODE
WRITE(TCONT,9995)
LEN = MAXTCO
TCONT(LEN) = NOCRLF
CALL SESC(FERR,TCONT,LEN)
CALL EXEC(2,CONWD2,TCONT,LEN)

```

C Write out specified number of blanks.

```

30 DO 40 I = 1,MAXREC
40 RECBUF(I) = 2H
CALL NOCR(FERR,RECBUF,LENCHR)
LENWDS = (LENCHR-1)/2+1
CALL EXEC(2,CONWD2,RECBUF,LENWDS)

```

C Now turn off inverse video.

```

CALL CLRTC
CALL CODE
WRITE(TCONT,9994)
LEN = MAXTCO
CALL SESC(FERR,TCONT,LEN)
TCONT(LEN) = NOCRLF
CALL EXEC(2,CONWD2,TCONT,LEN)

```

C Finally, end the definition of an unprotected field.

```

CALL CLRTC
CALL CODE
WRITE(TCONT,9993)
LEN = MAXTCO
CALL SESC(FERR,TCONT,LEN)
TCONT(LEN) = NOCRLF
CALL EXEC(2,CONWD2,TCONT,LEN)

```

10101 RETURN

```

9999 FORMAT(I1,1X,A1)
9998 FORMAT("(PREC5) Menu records out of order for menu ",I
- "; expected type 5 record, found instead: ",40A2)
9997 FORMAT("(PREC5) Illegal prompt attribute in menu ",I3,
- "; should be I or blank, found instead: ",40A2)
9996 FORMAT("E[")
9995 FORMAT("E&dB")
9994 FORMAT("E&a@")
9993 FORMAT("E]")

```

```

END
END$

```



SUBROUTINE PREC6(EOF,FERR)  
 -,1 NOV 83, V1.2

```

C*****
C*
C* Module name: PREC6 (PROCESS_RECORD_6)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 1 Nov 83
C*
C* Description:
C*
C* PREC6 takes the default values provided by type-6
C* records in menu (MD(NNN)) files and writes them to
C* the protected fields set up by PREC3. The default
C* values are truncated, if need be, to fit into the
C* unprotected default fields.
C*
C* Calling modules: DMENU.
C*
C* Modules called: CLRET, CLRRB, CLRTC, ERMSG, NOCR,
C* SESC.
C*
C* Data items input: CONWD2, CURMEN, LENCHR, MAXREC,
C* MAXTCO, MDDCB, NOCRLF, RECBUF.
C*
C* Data items output (changed): ERTEXT, FERR, TCONT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM2/MDDCB
COMMON /COM4/RECBUF

COMMON /COM22/CONWD2
COMMON /COM30/MAXREC

INTEGER MDDCB(144)
INTEGER RECBUF(40)

INTEGER CONWD2
INTEGER MAXREC

```

C Normal data item definitions.

```

INTEGER CURMEN,FLDLN,RECTYP
LOGICAL EOF,FERR

```

C Main program section.

```

C Read a type-4 record to get field length (note that
C file was positioned to first type-3 record by

```

```

C DMENU).

 CALL CLRRB
 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
 CALL CODE
 READ(RECBUF,9998) RECTYP,FLDLEN

C Read over type-5 record (must exist because passed edits
C of PREC1-PREC5).

 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)

C Read type-6 record to get default value.

 CALL CLRRB
 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
 CALL CODE
 READ(RECBUF,*) RECTYP
 IF (FLDLEN.EQ.0) GOTO 20

C Left justify prompting text by moving it over one word
C (erases "6," in first word).

 DO 10 I = 1,MAXREC-1
10 RECBUF(I) = RECBUF(I+1)

C Put an underscore at the end to inhibit CR/LF and write
C out text (with any needed spaces following the actual
C default value).

 CALL SCHAR(FERR,000000B,020000B,1,2*MAXREC,RECBUF)
 CALL NOCR(FERR,RECBUF,FLDLEN)
 LENWDS = (FLDLEN-1)/2+1
 CALL EXEC(2,CONWD2,RECBUF,LENWDS)

C Read next record (type-1). If end of file is
C encountered, exit normally.

20 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
 IF (LEN.LT.0) FEOF = .TRUE.
 IF (LEN.LT.0) GOTO 10101

C Read over next two records (type-2, type-3)

 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)
 CALL READF(MDDCB,IERR,RECBUF,MAXREC,LEN)

10101 RETURN

9999 FORMAT("(PREC6) Menu records out of order for menu "
- ,I3,"; expected type 6 record, found instead:",40A2)
9998 FORMAT(I1,I3)

```

END  
END\$

```

SUBROUTINE TMMENT(CURMEN,FBACK,FERR)
- ,4 NOV 83 V1.1
C*****
C*
C* Module name: TMMENT (TRAVERSE_MENU_TREE)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 4 Nov 83
C*
C* Description: Coordinates the traversal of the menu
C* tree. Arranges for proper menus to be displayed
C* according to the user's directions (BACKUP, etc.)
C*
C* Calling modules: SIOPT, GREQ.
C*
C* Modules called: CLRET, ERMSG, PARAM, SCHAR.
C*
C* Data items input: CURMEN, FBACK, PARBUF.
C*
C* Data items output: CURMEN, ERTEXT, FERR.
C*
C*****

```

C Common block data definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM7/PARBUF

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER MTNAM(3),MTBUF(4),FRMENU,TOMENU
INTEGER HOMENU,OPTNUM,CURMEN,OPTCNT
LOGICAL FERR,FBACK

```

C Main program section.

```

CALL CODE
WRITE(MTNAM,9999)

CALL OPEN(MDDCB,IERR,MTNAM)

```

C If file doesn't exist, write error message.

```

IF (IERR.GE.0) GOTO 10
FERR = .TRUE.
CALL CLRET
CALL CODE

```

```

 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)
 GOTO 10101

10 IF (FBACK) GOTO 80

C Move down the directory tree.

20 CALL READF(MDDCB,IERR,MTBUF,4,LEN)
 IF (LEN.GE.0) GOTO 30
 GOTO 10101

30 CALL CODE
 READ(MTBUF,*) FRMENU,TOMENU
 IF (FRMENU.NE.CURMEN) GOTO 20
 HOMENU = TOMENU
 CALL READF(MDDCB,IERR,MTBUF,4,LEN)
 IF (LEN.GE.0) GOTO 40
 CURMEN = HOMENU
 GOTO 10101

40 CALL CODE
 READ(MTBUF,*) FRMENU,TOMENU
 IF (FRMENU.EQ.CURMEN) GOTO 45
 CURMEN = HOMENU
 GOTO 10101

45 CALL RWNDF(MDDCB,IERR)
 CALL PARAM(1,FERR)
 IF (FERR) GOTO 10101
 CALL CODE
 READ(PARBUF,*) OPTNUM
 OPTCNT = 1

50 CALL READF(MDDCB,IERR,MTBUF,4,LEN)
 IF (LEN.GE.0) GOTO 60
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9995) OPTNUM,CURMEN
 CALL ERMSG(FERR)
 GOTO 10101

60 CALL CODE
 READ(MTBUF,*) FRMENU,TOMENU
 IF (FRMENU.NE.CURMEN) GOTO 50
 IF (OPTCNT.EQ.OPTNUM) GOTO 70
 OPTCNT = OPTCNT+1
 GOTO 50

70 CURMEN = TOMENU
 GOTO 10101

```

C Move backward (up) on the menu tree.

80 IF (CURMEN.EQ.1) GOTO 10101

90 CALL READF(MDDCB,IERR,MTBUF,4,LEN)  
IF (LEN.GE.0) GOTO 100  
FERR = .TRUE.  
CALL CLRET  
CALL CODE  
WRITE(ERTEXT,9996) CURMEN  
CALL ERMSG(FERR)  
GOTO 10101

100 CALL CODE  
READ(MTBUF,\*) FRMENU,TOMENU  
IF (TOMENU.NE.CURMEN) GOTO 90  
CURMEN = FRMENU

10101 RETURN

9999 FORMAT("MENUT ")  
9998 FORMAT(10X,"(TMENT) Menu tree file MENUT doesn't",  
- " exist.")  
9996 FORMAT(10X,"(TMENT) No backup entry in MENUT file ",  
- "for menu ",I3,".")  
9995 FORMAT(10X,"(TMENT) Option ",I3," entry doesn't ",  
- "exist for menu ",I3,".")

END  
END\$

SUBROUTINE AOPT(CURMEN,FERR,FEXIT,FRDEF,FREDIS)  
 - ,27 OCT 83 V1.2

```

C*****
C*
C* Module name: AOPT (ACCEPT_OPTIONS)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 27 Oct 83
C*
C* Description: AOPT accepts the user's terminal input
C* (options). Data is sent from the terminal in block
C* form; special characters separate the various data
C* fields.
C*
C* Calling modules: GOPT.
C*
C* Modules called: CLRRB, CLRSB, CLRTC, PARAM, SCHAR,
C* SESC, SIOPT.
C*
C* Data items input: CONWD1, CONWD2, CONWD3, CURMEN,
C* MAXSCR, MAXTCO, NOCRLF, PARBUF, RECBUF.
C*
C* Data items output (changed): FBACK, FDIREC, FERR,
C* FEXIT, FHELP, FRDEF, FREDIS, FSDEF, FSIOPT, MDDCB,
C* MENUSE, PARNUM, PARBUF, SCRDAT, TCONT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM2/MDDCB
COMMON /COM3/MENUSE
COMMON /COM4/RECBUF
COMMON /COM5/SCRDAT
COMMON /COM6/TCONT
COMMON /COM7/PARBUF

```

```

COMMON /COM21/CONWD1
COMMON /COM22/CONWD2
COMMON /COM23/CONWD3
COMMON /COM26/MAXSCR
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF

```

```

INTEGER MDDCB(144)
LOGICAL MENUSE(100)
INTEGER RECBUF(40)
INTEGER SCRDAT(500)
INTEGER TCONT(10)
INTEGER PARBUF(40)

```

```
INTEGER CONWD1
INTEGER CONWD2
INTEGER CONWD3
INTEGER MAXSCR
INTEGER MAXTCO
INTEGER NOCRLF
```

C Normal data item definitions.

```
INTEGER CURMEN
INTEGER CVDCB(144),CVNAM(3),FLDLN,ISIZE(2)
INTEGER MDNAM(3),PARNUM
LOGICAL FBACK,FDIREC,FERR,FEXIT,FHELP,FRDEF
LOGICAL FSDEF,FSIOPT,FSVREQ,FREDIS
```

C Initializations.

```
FSIOPT = .FALSE.
FHELP = .FALSE.
FBACK = .FALSE.
FSDEF = .FALSE.
FRDEF = .FALSE.
FSVREQ = .FALSE.
FEXIT = .FALSE.
FREDIS = .FALSE.
```

C Main program section.

C Accept the user's options in the form of a block of data  
C transferred from the terminal (which is operating in  
C block mode).

C First wait for the user to signal data entry is complete  
C (by hitting the "ENTER" key).

```
10 CALL CLRSD
 CALL EXEC(1,CONWD1,SCRDAT,MAXSCR)
```

C Check for function key usage. If valid function key, set  
C appropriate flag; otherwise, come back and wait for  
C valid response.

```
 IF (SCRDAT(1).NE.015560B) GOTO 20
 FHELP = .TRUE.
 FSIOPT = .TRUE.
 FREDIS = .TRUE.
 GOTO 80
20 IF (SCRDAT(1).NE.015561B) GOTO 30
 FBACK = .TRUE.
 FSIOPT = .TRUE.
 FREDIS = .TRUE.
 GOTO 80
30 IF (SCRDAT(1).NE.015562B) GOTO 40
```



```

 FSDEF = .TRUE.
 FSIOPT = .FALSE.
 FREDIS = .TRUE.
 GOTO 80
40 IF (SCRDAT(1).NE.015563B) GOTO 60
 FRDEF = .TRUE.
 FSIOPT = .FALSE.
 FREDIS = .TRUE.
 GOTO 10101
60 IF (SCRDAT(1).NE.015567B) GOTO 70
 FEXIT = .TRUE.
 FSIOPT = .FALSE.
 GOTO 10101
70 IF (IAND(SCRDAT(1),077400B).EQ.015400B) GOTO 10

C Home and backup cursor (so that all unprotected data
C on the screen will be transmitted), initiate block
C transfer.

80 CALL CLRTC
 CALL CODE
 WRITE(TCONT,9999)
 LEN = MAXTCO
 CALL SESC(FERR,TCONT,LEN)
 TCONT(LEN) = NOCRLF
 CALL EXEC(2,CONWD2,TCONT,LEN)

C Accept block transfer data into SCRDAT. May want to use
C larger buffer eventually...

 CALL CLRSD
 CALL EXEC(1,CONWD3,SCRDAT,MAXSCR)

C Call SIOPT if an immediate option has been requested.

 IF (FSIOPT) CALL SIOPT(CURMEN,FBACK,FDIREC,FERR,
- FEXIT,FHELP,FRDEF,FSDEF,FSVREQ)
 IF (FSIOPT) GOTO 10101

C Write out values to CV(NNN) file.
C Begin by opening files.

 MENUSE(CURMEN) = .TRUE.
 CALL CODE
 WRITE(MDNAM,9996) CURMEN
 CALL CODE
 WRITE(CVNAM,9995) CURMEN
 CALL SCHAR(FERR,1H ,1H0,1,5,MDNAM)
 CALL SCHAR(FERR,1H ,1H0,1,5,CVNAM)
 ISIZE(1) = 20
 ISIZE(2) = 0

```

C Get rid of (possibly) existing CV(NNN) file.

```
CALL CREAT(CVDCB,IERR,CVNAM,ISIZE,4,0,-20)
CALL OPEN(MDDCB,IERR,MDNAM,0,0,-20)
CALL OPEN(CVDCB,IERR,CVNAM,0,0,-20)
```

C Get to first type-4 record to read length of param.

```
PARNUM = 0
90 CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,40,LEN)
IF (LEN.LT.0) GOTO 120
CALL WRITF(CVDCB,IERR,RECBUF,LEN)
IF (RECBUF(1).NE.2H3,) GOTO 90
```

C Read field length from type-4 record.

```
CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,40,LEN)
CALL WRITF(CVDCB,IERR,RECBUF,LEN)
CALL CODE
READ(RECBUF,9993) I,FLDLEN
```

C Skip type-5 record.

```
CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,40,LEN)
CALL WRITF(CVDCB,IERR,RECBUF,LEN)
```

C Read type-6 record.

```
CALL CLRRB
CALL READF(MDDCB,IERR,RECBUF,40,LEN)
IF (FLDLEN.GT.0) GOTO 110
CALL WRITF(CVDCB,IERR,RECBUF,LEN)
GOTO 90
```

C There is a default value associated with this  
C type-6 record (type-3 was a true prompt).

```
110 PARNUM = PARNUM+1
IF (FLDLEN.GT.38) FLDLEN = 38
CALL PARAM(PARNUM,FERR)
IF (FERR) GOTO 10101
CALL CODE
WRITE(RECBUF,9994) (PARBUF(I),I=1,(FLDLEN-1)/2+1)
CALL WRITF(CVDCB,IERR,RECBUF,(FLDLEN-1)/2+2)
GOTO 90
```

C If the user asked for the current values to become  
C new defaults, copy the current-value file into  
C the default (MD(NNN)) file.

```

120 IF (.NOT.FSDEF) GOTO 10101
 CALL RWNDF(MDDCB)
 CALL RWNDF(CVDCB)
130 CALL CLRRB
 CALL READF(CVDCB,IERR,RECBUF,40,LEN)
 IF (LEN.LT.0) GOTO 10101
 CALL WRITF(MDDCB,IERR,RECBUF,LEN)
 GOTO 130

10101 CALL CLOSE(MDDCB)
 CALL CLOSE(CVDCB)

9999 FORMAT("EHEDEd")
9998 FORMAT("EXE&k0 BEH")
9997 FORMAT(5(8(2X,O6),/))
C997 FORMAT(2(20A2,/))
9996 FORMAT("MD",I3," ")
9995 FORMAT("CV",I3," ")
9994 FORMAT("6"," ",39A2)
9993 FORMAT(I1,1X,I3)

```

```

END
END$

```

```

SUBROUTINE SIOPT(CURMEN,FBACK,FDIREC,FERR,
- FEXIT,FHELP,FRDEF,FSDEF,FSVREQ)
-,15 OCT 83, V1.2

```

```

C*****
C*
C* Module name: SIOPT (SATISFY_IMMEDIATE_OPTIONS)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 15 Oct 83
C*
C* Description: Manages modules responsible for satis-
C* fying immediate user options. Immediate user options
C* are those which may be satisfied without having to
C* perform any data acquisition, processing, or display
C* (e.g. help, saving new defaults, etc.)
C*
C* Calling modules: AOPT.
C*
C* Modules called: DIREC, HELP, SVREQ, TMENT.
C*
C* Data items input: CURMEN, FBACK, FDIREC, FERR, FEXIT,
C* FHELP, FRDEF, FSDEF, FSVREQ.
C*
C* Data items output (changed): FERR.
C*
C*****

```

C Normal data item definitions.

```

INTEGER CURMEN,DIRDES,DIRTYP
LOGICAL FBACK,FDIREC,FERR,FEXIT,FHELP
LOGICAL FRDEF,FSDEF,FSVREQ

```

C Main section.

```

 IF (.NOT.FBACK) GOTO 10
 CALL TMENT(CURMEN,FBACK,FERR)
 GOTO 10101
10 IF (.NOT.FDIREC) GOTO 20
C CALL DIREC(DIRDES,DIRTYP,FERR)
 GOTO 10101
20 IF (.NOT.FHELP) GOTO 30
 CALL HELP(CURMEN,FERR)
 GOTO 10101
30 IF (.NOT.FSVREQ) GOTO 40
C CALL SVREQ(REQFIL,FERR)
 GOTO 10101
40 CONTINUE

10101 RETURN

```

END  
END\$

```

 SUBROUTINE HELP(CURMEN,FERR)
 - , 15 OCT 83 V1.1
C*****
C*
C* Module name: HELP
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 15 Oct 83
C*
C* Description: Provides help messages to the user when
C* the user requests them by pressing the HELP special
C* function key on the keyboard.
C*
C* Calling modules: SIOPT.
C*
C* Modules called: BORDR, CLRET, CLRRB, CURSR, ERMSG.
C*
C* Data items input: CONWD1, CONWD2, CURMEN, FERR,
C* MAXREC, NOCRLF, RECBUF.
C*
C* Data items output: ERTEXT, FERR.
C*
C*****

```

C Common block definitions.

```

 COMMON /COM1/ERTEXT
 COMMON /COM4/RECBUF

```

```

 COMMON /COM21/CONWD1
 COMMON /COM22/CONWD2
 COMMON /COM30/MAXREC
 COMMON /COM32/NOCRLF
 INTEGER ERTEXT(80)
 INTEGER RECBUF(40)
 INTEGER CONWD1
 INTEGER CONWD2
 INTEGER MAXREC
 INTEGER NOCRLF

```

C Normal data item definitions.

```

 INTEGER CURMEN,HLPDCB(144),HLPNAM(3),LINENO,PAGEDE
 INTEGER PAGENO,PGLENG
 LOGICAL FERR

```

C Initializations.

```

 PGLENG = 20
 PAGENO = 1
 PAGEDE = 1

```

C Main program section.

C Open help file. Tell user "No help available" if error.

```
CALL CODE
WRITE(HLPNAM,9999) CURMEN
CALL SCHAR(FERR,1H ,1H0,1,6,HLPNAM)
CALL OPEN(HLPDCB,IERR,HLPNAM)
IF (IERR.GE.0) GOTO 10
CALL CLRET
CALL CODE
WRITE(ERTEXT,9998) (HLPNAM(I),I=1,3)
CALL ERMSG(FERR)
GOTO 10101
```

C Draw border for new page.

```
10 CALL BORDR(0,FERR)
CALL CURSR(1,0,FERR)
LINENO = 0
```

C Write out PGLENG lines (after positioning cursor)

```
20 CALL CLRRB
CALL READF(HLPDCB,IERR,RECBUF,MAXREC,LEN)
IF (LEN.GE.0) GOTO 30
```

C If EOF, user may back up or exit. If the user hits  
C the EXIT key, control will return to the  
C calling program. The BACKUP key displays the previous  
C page. Anything else is ignored.

```
CALL CURSR(23,0)
CALL CLRET
CALL CODE
WRITE(ERTEXT,9996)
ERTEXT(40) = NOCRLF
CALL EXEC(2,CONWD2,ERTEXT,40)
23 CALL EXEC(1,CONWD1,RECBUF,MAXREC)
IF (RECBUF(1).EQ.015561B) GOTO 25
IF (RECBUF(1).EQ.015567B) GOTO 10101
GOTO 23
25 PAGEDE = PAGEDE-1
IF (PAGEDE.LT.1) PAGEDE = 1
PAGEDE = 1
CALL RWNDF(HLPDCB)
GOTO 10
```

C EOF has not been reached. If the page being read is the  
C page the user wants, print out the record.

```
30 IF (PAGEDE.EQ.PAGEDE) WRITE(1,9997) (RECBUF(I),I=1,39)
LINENO = LINENO+1
```

C Has end of page been reached? If so, wait for the user's  
C instruction to backup, proceed, or exit.

```
 IF (LINENO.LT.PGLENG) GOTO 20
 LINENO = 0
 IF (PAGENO.NE.PAGEDE) GOTO 60
40 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9995)
 ERTEXT(40) = NOCRLF
 CALL CURSR(23,0,FERR)
 CALL EXEC(2,CONWD2,ERTEXT,40)
 CALL EXEC(1,CONWD1,RECBUF,MAXREC)
 IF (RECBUF(1).EQ.015567B) GOTO 10101
 IF (RECBUF(1).NE.015561B) GOTO 50
 CALL RWNDF(HLPDCB)
 PAGEDE = PAGEDE-1
 IF (PAGEDE.LT.1) PAGEDE = 1
 PAGENO = PAGENO-1
 IF (PAGENO.LT.1) PAGENO = 1
 GOTO 10
50 IF (IAND(RECBUF(1),077400B).EQ.015400B) GOTO 40
 PAGEDE = PAGEDE+1

60 PAGENO = PAGENO+1
 GOTO 10

10101 CALL CLOSE(HLPDCB)
 RETURN

9999 FORMAT("HLP",I3)
9998 FORMAT(8X,"No help available for this menu; file ",
- 3A2," doesn't exist.")
9997 FORMAT(39A2)
9996 FORMAT(13X,"(Hit EXIT to exit, BACKUP to backup",
- " to previous menu)")
9995 FORMAT(2X,"(Hit ENTER for next page, BACKUP for",
- " previous page, EXIT to return to menu)")

 END
 END$
```



```

SUBROUTINE EOPT(ACQNO,CURMEN,DISNO,FDIREC,FERR,FEXE,
- FGEDIT,FREREQ,FSVREQ,PRONO)
- ,28 OCT 83 V1.2

```

```

C*****
C*
C* Name: EOPT (EDIT_OPTIONS)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 28 Oct 83
C*
C* Description:
C*
C* EOPT calls subroutines (EM(NNN)) which edit the
C* user's options. There is one such subroutine for
C* each menu.
C*
C* Calling module: GREQ.
C*
C* Modules called: EM001, EM001,...
C*
C* Data items input: ACQNO, CURMEN, DISNO, FDIREC, FERR,
C* FEXE, FGEDIT, FREREQ, FSVREQ, PRONO.
C*
C* Data items output (changed): FGEDIT.
C*
C*****

```

C Normal data item definitions.

```

INTEGER ACQNO,CURMEN,DISNO,PRONO(10)
LOGICAL FDIREC,FERR,FEXE,FGEDIT,FREREQ,FSVREQ

```

C Initializations.

```

FGEDIT = .TRUE.

```

C Main section.

```

IF (CURMEN.NE.1) GOTO 2
1 CALL EM001(FDIREC,FERR,FEXE,FGEDIT,FREREQ,FSVREQ)
 GOTO 10101

2 IF (CURMEN.NE.2) GOTO 3
 CALL EM002(ACQNO,FERR,FGEDIT)
 GOTO 10101

3 IF (CURMEN.NE.3) GOTO 4
 CALL EM003(FERR,FGEDIT)
 GOTO 10101

```

4 IF (CURMEN.NE.4) GOTO 5  
CALL EM004(DISNO,FERR,FGEDIT)  
GOTO 10101

5 IF (CURMEN.NE.5) GOTO 6  
CALL EM005(FERR,FGEDIT)  
GOTO 10101

6 IF (CURMEN.NE.6) GOTO 7  
CALL EM006(FERR,FGEDIT)  
GOTO 10101

7 IF (CURMEN.NE.7) GOTO 8  
CALL EM007(FERR,FGEDIT,PRONO)  
GOTO 10101

8 IF (CURMEN.NE.8) GOTO 9  
CALL EM008(FERR,FGEDIT,PRONO)  
GOTO 10101

9 IF (CURMEN.NE.9) GOTO 10  
CALL EM009(FERR,FGEDIT)  
GOTO 10101

10 IF (CURMEN.NE.10) GOTO 10101  
CALL EM010(FERR,FGEDIT)  
GOTO 10101

10101 RETURN

END  
END\$

```

SUBROUTINE EM001(FDIREC,FERR,FEXE,FGEDIT,
- FREREQ,FSVREQ),31 OCT 83 V1.1

```

```

C*****
C*
C* Module name: EM001
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 31 Oct 83
C*
C* Description: Edits the user-supplied options from menu
C* number 1.
C*
C* Calling modules: EOPT.
C*
C* Modules called: CLRET, ERMSG.
C*
C* Data items input: FERR, SCRDAT.
C*
C* Data items output: ERTEXT, FDIREC, FEXE, FGEDIT,
C* FREREQ, FSVREQ.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER CHOICE
LOGICAL FDIREC,FERR,FEXE,FGEDIT,FREREQ,FSVREQ

```

C Initializations.

```

CALL PARAM(1,FERR)
CALL CODE
READ(PARBUF,9998) CHOICE
FGEDIT = .TRUE.

```

C Main program section.

```

 IF (CHOICE.NE.1) GOTO 10
 GOTO 10101
10 IF (CHOICE.NE.2) GOTO 20
 GOTO 10101
20 IF (CHOICE.NE.3) GOTO 60
 GOTO 10101

```

```

30 IF (CHOICE.NE.4) GOTO 40
 FSVREQ = .TRUE.
 GOTO 10101
40 IF (CHOICE.NE.5) GOTO 50
 FREREQ = .TRUE.
 GOTO 10101
50 IF (CHOICE.NE.6) GOTO 60
 FDIREC = .TRUE.
 GOTO 10101
60 IF (CHOICE.NE.7) GOTO 70
 FEFE = .TRUE.
 GOTO 10101

70 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999)
 CALL ERMSG(FERR)

10101 RETURN

9999 FORMAT(10X,"(EM001) Current valid options are",
- " 1, 2, 3, and 7")
9998 FORMAT(I1)

 END
 END$

```

SUBROUTINE EM002(ACQNO,FERR,FGEDIT),31 OCT 83 V1.1

```
C*****
C*
C* Module name: EM002 (EDIT_MENU_2)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 31 Oct 83
C*
C* Description: Edits the user-supplied options from
C* menu 2.
C*
C* Calling modules: EOPT.
C*
C* Modules called: CLRET, ERMSG.
C*
C* Data items input: SCRDAT.
C*
C* Data items output: ACQNO, ERTEXT, FERR, FGEDIT.
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF
```

```
INTEGER ERTEXT(80)
INTEGER PARBUF(40)
```

C Normal data item definitions.

```
INTEGER ACQNO,CHOICE
LOGICAL FERR,FGEDIT
```

C Initializations.

```
FGEDIT = .TRUE.
CALL PARAM(1,FERR)
CALL CODE
READ(PARBUF,9998) CHOICE
```

C Main program section.

```
C Accept only valid options. Subtract one from the
C option selected so that it corresponds to the
C module number of the desired acquisition module.
```

```
IF (CHOICE.NE.2) GOTO 10
ACQNO = 1
GOTO 10101
```

```
10 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999)
 CALL ERMSG(FERR)

10101 RETURN

9999 FORMAT(10X,"(EM002) Only option 2 is currently ",
- "supported.")
9998 FORMAT(I1)

 END
 END$
```

SUBROUTINE EM003(FERR,FGEDIT),31 OCT 83,V1.1

```
C*****
C*
C* Module name: EM003 (EDIT_MENU_3)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 31 Oct 83
C*
C* Description: Edits the user-supplied options from
C* menu 3.
C*
C* Calling modules: EOPT.
C*
C* Modules called: CLRET, ERMSG.
C*
C* Data items input: SCRDAT.
C*
C* Data items output: ERTEXT, FERR, FGEDIT, PRONO.
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF
```

```
INTEGER ERTEXT(80)
INTEGER PARBUF(40)
```

C Normal data item definitions.

```
INTEGER CHOICE
LOGICAL FERR,FGEDIT
```

C Initializations.

```
FGEDIT = .TRUE.
CALL PARAM(1,FERR)
CALL CODE
READ(PARBUF,9998) CHOICE
```

C Main program section.

```
IF ((CHOICE.EQ.4).OR.(CHOICE.EQ.6)) GOTO 10101
FGEDIT = .FALSE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999)
CALL ERMSG(FERR)
GOTO 10101
```

10101 RETURN

9999 FORMAT(10X,"(EM003) Only options 4 and 6 are ",  
- "currently supported.")

9998 FORMAT(11)

END

END\$



SUBROUTINE EM004(DISNO,FERR,FGEDIT),31 OCT 83 V1.1

```
C*****
C*
C* Module name: EM004 (EDIT_MENU_4)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 31 Oct 83
C*
C* Description: Edits the user-supplied options from
C* menu 4.
C*
C* Calling modules: EOPT.
C*
C* Modules called: CLRET, ERMSG.
C*
C* Data items input: SCRDAT.
C*
C* Data items output: DISNO, ERTEXT, FERR, FGEDIT.
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF
```

```
INTEGER ERTEXT(80)
INTEGER PARBUF(40)
```

C Normal data item definitions.

```
INTEGER CHOICE,DISNO
LOGICAL FERR,FGEDIT
```

C Initializations.

```
FGEDIT = .TRUE.
CALL PARAM(1,FERR)
CALL CODE
READ(PARBUF,9998)CHOICE
```

C Main program section.

```
IF (CHOICE.EQ.2) GOTO 10
FGEDIT = .FALSE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999)
CALL ERMSG(FERR)
GOTO 10101
```

```
10 DISNO = 1
10101 RETURN
9999 FORMAT(10X,"(EM004) Only option 2 is currently ",
- "supported.")
9998 FORMAT(I1)

 END
 END$
```

SUBROUTINE EM005(FERR,FGEDIT),8 NOV 83 V1.1

```

C*****
C*
C* Module name: EM005 (EDIT_MENU_5)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description:
C*
C* EM005 edits menu 5 values, returning FGEDIT = FALSE
C* if any are found unacceptable.
C*
C* Calling modules: EOPT.
C*
C* Modules called: CLRET, ERMSG, PARAM.
C*
C* Data items input: PARBUF.
C*
C* Data items output: FERR, FGEDIT, PARNUM, ERTEXT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER IDCB(144),NAME(3),PARNUM

LOGICAL FERR,FGEDIT

```

C Main program section.

```

 PARNUM = 1
 CALL PARAM(PARNUM,FERR)
 DO 10 I = 1,3
10 NAME(I) = PARBUF(I)
 CALL OPEN(IDCB,IERR,NAME)
 IF (IERR.GE.0) GOTO 20
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) (NAME(I),I=1,3)
 CALL ERMSG(FERR)
 GOTO 10101

```

```

20 PARNUM = 2
 CALL PARAM(PARNUM,FERR)
 IF ((PARBUF(1).EQ.2HR).OR.(PARBUF(1).EQ.2HC))
- GOTO 10101
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)

10101 CALL CLOSE(IDCBI)
 RETURN

9999 FORMAT(5X,"(EM005) File ",3A2," not found.")
9998 FORMAT(5X,"(EM005) R and C are the only data ",
- "types that are recognized.")

 END
 END$

```

SUBROUTINE EM006(FERR,FGEDIT),8 NOV 83 V1.1

```

C*****
C*
C* Module name: EM006 (EDIT_MENU_6)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description: Edits user-supplied options to menu 6.
C*
C* Calling modules: EOPT.
C*
C* Modules called: NUMER, ERMSG, PARAM.
C*
C* Data items input: PARBUF.
C*
C* Data items output: FERR, FGEDIT, PARNUM, ERTEXT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER BLOCKS,PARNUM

LOGICAL FERR,FGEDIT,NUMER

EXTERNAL NUMER

```

C Main program section.

```

 PARNUM = 1
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(4)) GOTO 10
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999)
 CALL ERMSG(FERR)
 GOTO 10101
10 PARNUM = 2
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(4)) GOTO 15

```

```

 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)
 GOTO 10101

15 CALL CODE
 READ(PARBUF,9988) BLOCKS
 IF (BLOCKS.NE.9999) GOTO 20
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9997)
 CALL ERMSG(FERR)
 GOTO 10101

20 PARNUM = 3
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(6)) GOTO 30
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9996)
 CALL ERMSG(FERR)
 GOTO 10101

30 PARNUM = 4
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(5)) GOTO 40
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9995)
 CALL ERMSG(FERR)
 GOTO 10101

40 PARNUM = 5
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(5)) GOTO 50
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9994)
 CALL ERMSG(FERR)
 GOTO 10101

50 PARNUM = 6
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(3)) GOTO 60
 FGEDIT = .FALSE.
 CALL CLRET

```

```

 CALL CODE
 WRITE(ERTEXT,9993)
 CALL ERMSG(FERR)
 GOTO 10101

60 PARNUM = 7
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(6)) GOTO 70
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9992)
 CALL ERMSG(FERR)
 GOTO 10101

70 PARNUM = 8
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(5)) GOTO 80
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9991)
 CALL ERMSG(FERR)
 GOTO 10101

80 PARNUM = 9
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(5)) GOTO 90
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9990)
 CALL ERMSG(FERR)
 GOTO 10101

90 PARNUM = 10
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(3)) GOTO 100
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9989)
 CALL ERMSG(FERR)
 GOTO 10101

100 FGEDIT = .TRUE.

10101 RETURN

9999 FORMAT(5X,"(EM006) Number of data points should ",
- "be numeric.")
9998 FORMAT(5X,"(EM006) Number of blocks should be",

```

```

- " numeric.")
9997 FORMAT(5X,"(EM006) Continuous data generation is",
- " not supported at this time.")
9996 FORMAT(5X,"(EM006) Scaling factor in first term",
- " should be numeric.")
9995 FORMAT(5X,"(EM006) Sampling frequency in first term",
- " should be numeric.")
9994 FORMAT(5X,"(EM006) Sinusoid frequency in first",
- " term should be numeric.")
9993 FORMAT(5X,"(EM006) Phase shift in first term "
- ",should be numeric.")
9992 FORMAT(5X,"(EM006) Scaling factor in second ",
- "term should be numeric.")
9991 FORMAT(5X,"(EM006) Sampling frequency in second",
- " term should be numeric.")
9990 FORMAT(5X,"(EM006) Sinusoid frequency in",
- " second term should be numeric.")
9989 FORMAT(5X,"(EM006) Phase shift in second term ",
- "should be numeric.")
9988 FORMAT(I4)

```

```

END
END$

```



SUBROUTINE EM007(FERR,FGEDIT,PRONO)  
- ,19 NOV 83 V1.1

```
C*****
C*
C* Module name: EM007 (EDIT_MENU_7)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 19 Nov 83
C*
C* Description: Edits user-supplied options to menu 7.
C*
C* Calling modules: EOPT.
C*
C* Modules called: ERMSG, NUMER, PARAM.
C*
C* Data items input: PARBUF.
C*
C* Data items output: FERR, FGEDIT, ERTEXT, PARNUM.
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF
```

```
INTEGER ERTEXT(80)
INTEGER PARBUF(40)
```

C Normal data item definitions.

```
INTEGER BLOCKS,PARNUM,POINTS
INTEGER PRONO(10),PRONUM
```

```
LOGICAL FERR,FGEDIT,NUMER
```

```
EXTERNAL NUMER
```

C Main program section.

```
PARNUM = 1
CALL PARAM(PARNUM,FERR)
IF (NUMER(4)) GOTO 10
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999)
 CALL ERMSG(FERR)
 GOTO 10101
```

10 CALL CODE

```

READ(PARBUF,9994) POINTS
J = 2
DO 20 I = 1,15
 IF (POINTS.EQ.J) GOTO 30
 J = J*2
20 CONTINUE
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)
 GOTO 10101

30 PARNUM = 2
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(4)) GOTO 40
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9997)
 CALL ERMSG(FERR)
 GOTO 10101

40 CALL CODE
 READ(PARBUF,9994) BLOCKS
 IF (BLOCKS.GE.1) GOTO 50
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9996)
 CALL ERMSG(FERR)
 GOTO 10101

50 IF (BLOCKS.NE.9999) GOTO 60
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9995)
 CALL ERMSG(FERR)
 GOTO 10101

60 PARNUM = 3
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(2)) GOTO 70
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9993)
 CALL ERMSG(FERR)
 GOTO 10101

70 CALL CODE

```

```

 READ(PARBUF,9994) PRONUM
 IF ((PRONUM.GE.0).AND.(PRONUM.LE.2)) GOTO 80
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9993)
 CALL ERMSG(FERR)
 GOTO 10101

80 IF (PRONUM.GT.0) GOTO 90
 IF (PRONO(1).EQ.4) PRONO(1) = 0
 IF (PRONO(2).EQ.4) PRONO(2) = 0
 GOTO 10101

90 PRONO(PRONUM) = 4

10101 RETURN

9999 FORMAT(5X,"(EM007) Number of data points per block",
- " should be numeric.")
9998 FORMAT(5X,"(EM007) Number of data points per block",
- " should be a positive power of 2.")
9997 FORMAT(5X,"(EM007) Number of blocks should be",
- " numeric.")
9996 FORMAT(5X,"(EM007) Number of blocks should be",
- " a positive integer.")
9995 FORMAT(5X,"(EM007) Continuous processing is not",
- " yet supported.")
9994 FORMAT(I4)
9993 FORMAT(5X,"(EM007) Process number must be ",
- "between 0 and 2.")

 END
 END$

```

SUBROUTINE EM008(FERR,FGEDIT,PRONO)  
- ,19 NOV 83 V1.1

```
C*****
C*
C* Module name: EM008 (EDIT_MENU_8)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 19 Nov 83
C*
C* Description: Edits the user-supplied options for
C* menu 9.
C*
C* Calling modules: EOPT.
C*
C* Modules called: ERMSG, NUMER, PARAM.
C*
C* Data items input: PARBUF.
C*
C* Data items output: FERR, FGEDIT, ERTEXT, PARNUM.
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF
```

```
INTEGER ERTEXT(80)
INTEGER PARBUF(40)
```

C Normal data item definitions.

```
INTEGER BLOCKS,PARNUM,POINTS
INTEGER PRONO(10),PRONUM
```

```
LOGICAL FERR,FGEDIT,NUMER
```

```
EXTERNAL NUMER
```

C Main program section.

```
PARNUM = 1
CALL PARAM(PARNUM,FERR)
IF (NUMER(4)) GOTO 10
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)
 GOTO 10101
```

```

10 CALL CODE
 READ(PARBUF,9994) POINTS
 J = 2
 DO 20 I = 1,15
 IF (POINTS.EQ.J) GOTO 30
 J = J*2
20 CONTINUE
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)
 GOTO 10101

30 PARNUM = 2
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(4)) GOTO 40
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9996)
 CALL ERMSG(FERR)
 GOTO 10101

40 CALL CODE
 READ(PARBUF,9994) BLOCKS
 IF (BLOCKS.GE.1) GOTO 50
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9996)
 CALL ERMSG(FERR)
 GOTO 10101

50 IF (BLOCKS.NE.9999) GOTO 60
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9995)
 CALL ERMSG(FERR)
 GOTO 10101

60 PARNUM = 3
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(2)) GOTO 70
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9993)
 CALL ERMSG(FERR)
 GOTO 10101

```

```

70 CALL CODE
 READ(PARBUF,9994) PRONUM
 IF ((PRONUM.GE.0).AND.(PRONUM.LE.2)) GOTO 80
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9993)
 CALL ERMSG(FERR)
 GOTO 10101

80 IF (PRONUM.GT.0) GOTO 90
 IF (PRONO(1).EQ.6) PRONO(1) = 0
 IF (PRONO(2).EQ.6) PRONO(2) = 0
 GOTO 10101

90 PRONO(PRONUM) = 6

10101 RETURN

9998 FORMAT(5X,"(EM008) Number of data points per block",
- " should be a positive power of 2.")
9996 FORMAT(5X,"(EM008) Number of blocks should be",
- " a positive integer.")
9995 FORMAT(5X,"(EM008) Continuous processing is not",
- " yet supported.")
9994 FORMAT(I4)
9993 FORMAT(5X,"(EM008) Process number must be ",
- "either 1 or 2.")

 END
 END$

```

SUBROUTINE EM009(FERR,FGEDIT),8 NOV 83 V1.1

```

C*****
C*
C* Module name: EM009 (EDIT_MENU_9)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description: Edits user-supplied options for menu 9.
C*
C* Calling modules: EOPT.
C*
C* Modules called: ERMSG, PARAM.
C*
C* Data items input: PARBUF.
C*
C* Data items output: ERTEXT, FERR, FGEDIT, PARNUM.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER PARNUM

LOGICAL FERR,FGEDIT

```

C Main program section.

```

PARNUM = 2
CALL PARAM(PARNUM,FERR)
IF ((PARBUF(1).EQ.2HR).OR.(PARBUF(1).EQ.2HC))
- GOTO 10101
FGEDIT = .FALSE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999)
CALL ERMSG(FERR)

```

10101 RETURN

```

9999 FORMAT(5X,"(EM009) Only real (R) and complex (C)",
- " data types are recognized.")

```

END  
END\$



SUBROUTINE EM010(FERR,FGEDIT),8 NOV 83 V1.1

```

C*****
C*
C* Module name: EM010 (EDIT_MENU_10)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description: Edits user-supplied options for menu 10.
C*
C* Calling modules: EOPT.
C*
C* Modules called: ERMSG, NUMER, PARAM.
C*
C* Data items input: PARBUF.
C*
C* Data items output: ERTEXT, FERR, FGEDIT, PARNUM.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER BLOCKS,PARNUM,POINTS

LOGICAL FERR,FGEDIT,NUMER

EXTERNAL NUMER

```

C Main program section.

```

PARNUM = 1
CALL PARAM(PARNUM,FERR)
IF (NUMER(4)) GOTO 10
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999)
 CALL ERMSG(FERR)
 GOTO 10101

10 CALL CODE
 READ(PARBUF,9994) POINTS
 J = 2

```

```

DO 20 I = 1,15
 IF (POINTS.EQ.J) GOTO 30
 J = J*2
20 CONTINUE
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)
 GOTO 10101

30 PARNUM = 2
 CALL PARAM(PARNUM,FERR)
 IF (NUMER(4)) GOTO 40
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9997)
 CALL ERMSG(FERR)
 GOTO 10101

40 PARNUM = 3
 CALL PARAM(PARNUM,FERR)
 IF ((PARBUF(1).EQ.2HR).OR.(PARBUF(1).EQ.2HC))
- GOTO 50
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9996)
 CALL ERMSG(FERR)
 GOTO 10101

50 PARNUM = 4
 CALL PARAM(PARNUM,FERR)
 IF ((PARBUF(1).EQ.2HR).OR.(PARBUF(1).EQ.2HI).OR.
- (PARBUF(1).EQ.2HM)) GOTO 10101
 FGEDIT = .FALSE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9995)
 CALL ERMSG(FERR)
 GOTO 10101

```

10101 RETURN

```

9999 FORMAT(5X,"(EM010) Number of data points per",
- " block should be numeric.")
9998 FORMAT(5X,"(EM010) Number of data points per",
- " block should be a positive power of 2.")
9997 FORMAT(5X,"(EM010) Number of blocks should ",
- "be a positive integer.")
9996 FORMAT(5X,"(EM010) Recognized data types are ",

```

- "real (R) and complex (C)."  
9995 FORMAT(5X,"(EM010) Only real (R), imaginary (I), ",  
- "and magnitude (M) are permitted for part.")  
9994 FORMAT(I4)

END  
END\$

SUBROUTINE FREQ(ACQNO,DISNO,FCONS,PRONO,FERR)  
 -,31 OCT 83, V1.2

```

C*****
C*
C* Module name: FREQ (FORMAT_REQUEST)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 31 Oct 83
C*
C* Description:
C*
C* FREQ manages modules which format the user's
C* so that they are compatible with what the system's
C* application programs expect. Data is taken from
C* CV/(NNN) files and formed into ACQ(XXX) records.
C*
C* Calling modules: GREQ.
C*
C* Modules called: FA001, FA002,...FD001, FD002,...
C* FP001, FP002,..., CLRET, ERMSG.
C*
C* Data items input: ACQNO, DISNO, PRONO.
C*
C* Data items output (changed): ERTEXT, FERR.
C*
C*****

```

C Common block definitions.

COMMON /COM1/ERTEXT

INTEGER ERTEXT(80)

C Normal data item definitions.

INTEGER ACQNO,DISNO,PRONO(10)  
 LOGICAL FCONS,FERR

C Main program section.

```

C10 IF (ACQNO.NE.0) GOTO 20
C CALL FA000(FERR)
C IF (FERR) GOTO 10101
C GOTO 50

20 IF (ACQNO.NE.1) GOTO 30
 CALL FA001(FERR)
 IF (FERR) GOTO 10101
 GOTO 50

30 IF (ACQNO.NE.2) GOTO 40

```

C       CALL FA002(FERR)  
C       IF (FERR) GOTO 10101  
C       GOTO 50

C Error.

40       FERR = .TRUE.  
          CALL CLRET  
          CALL CODE  
          WRITE(ERTEXT,9999)  
          CALL ERMSG(FERR)  
          GOTO 10101

50       IF ((PRONO(1).EQ.0).AND.(PRONO(2).EQ.0)) GOTO 80  
          IF ((PRONO(1).NE.4).AND.(PRONO(2).NE.4)) GOTO 60  
          CALL FP004(FERR)  
          IF (FERR) GOTO 10101  
          GOTO 80

60       IF ((PRONO(1).NE.6).AND.(PRONO(2).NE.6)) GOTO 60  
          CALL FP006(FERR)  
          IF (FERR) GOTO 10101  
          GOTO 80

C Error.

70       FERR = .TRUE.  
          CALL CLRET  
          CALL CODE  
          WRITE(ERTEXT,9998)  
          CALL ERMSG(FERR)  
          GOTO 10101

80       IF (DISNO.NE.0) GOTO 90  
C       CALL FD000(FERR)  
C       IF (FERR) GOTO 10101  
C       GOTO 10101

90       IF (DISNO.NE.1) GOTO 100  
          CALL FD001(FERR)  
          IF (FERR) GOTO 10101  
          GOTO 10101

100       IF (DISNO.NE.2) GOTO 110  
C       CALL FD002(FERR)  
C       IF (FERR) GOTO 10101  
C       GOTO 10101

C Error: invalid display request.

110       FERR = .TRUE.  
          CALL CLRET

```
CALL CODE
WRITE(ERTEXT,9997)
CALL ERMSG(FERR)
GOTO 10101
```

```
10101 RETURN
```

```
9999 FORMAT(10X,"(FREQ) Acquisition request number must",
- " be between 0 and 2.")
9998 FORMAT(10X,"(FREQ) Processing request number must",
- " be either 4 or 6.")
9997 FORMAT(10X,"(FREQ) Only display option 2 is ",
- "currently supported.")
```

```
END
END$
```

# SUBROUTINE FA001(FERR),8 NOV 83 V1.1

```

C*****
C*
C* Module name: FA001 (FORMAT_ACQUISITION_REQUEST_1)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description: Formats the request provided to AD001
C* to guide its actions when it is called upon to
C* generate data.
C*
C* Calling modules: FREQ.
C*
C* Modules called: CLRET, ERMSG.
C*
C* Data items input: PARBUF.
C*
C* Data items output: ERTEXT, FERR, MDDCB, PARBUF.
C*
C*****

```

## C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM3/MENUSE
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
LOGICAL MENUSE(100)
INTEGER PARBUF(40)

```

## C Normal data item definitions.

```

INTEGER ARDCB(144),ARNAM(3),ISIZE(2),MDNAM(3)
INTEGER NUMPAR,PARNUM,CVNAM(3)

```

```

LOGICAL FERR

```

## C Initializations.

```

DATA MDNAM/2HMD,2H00,2H6 /,NUMPAR/10/
DATA ARNAM/2HAR,2HEQ,2H /,ISIZE/20,0/
DATA CVNAM/2HCV,2H00,2H6 /

```

## C Main program section.

```

IF (MENUSE(6)) GOTO 5
CALL OPEN(MDDCB,IERR,MDNAM)

```

```

 GOTO 7
5 CALL OPEN(MDDCB,IERR,CVNAM)
7 IF (IERR.GE.0) GOTO 10
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) (MDNAM(I),I=1,3)
 CALL ERMSG(FERR)
 GOTO 10101

10 CALL OPEN(ARDCB,IERR,ARNAM)
 IF (IERR.LT.0) CALL CREAT(ARDCB,IERR,ARNAM,ISIZE,4)

20 DO 30 PARNUM = 1,NUMPAR
 CALL FPAR(PARNUM,FERR)
 IF (FERR) GOTO 10101
 CALL WRITF(ARDCB,IERR,PARBUF,40)
30 CONTINUE

10101 CALL CLOSE(ARDCB)
 CALL CLOSE(MDDCB)
 RETURN

9999 FORMAT(5X,"(FA001) Can't open MD006.")

 END
 END$

```



SUBROUTINE FP004(FERR),8 NOV 83 V1.1

```

C*****
C*
C* Module name: FP004 (FORMAT_PROCESSING_REQUEST_4)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description: Formats the request which is provided to
C* PD004 to guide its actions when it is called upon to
C* perform an FFT.
C*
C* Calling modules: FREQ.
C*
C* Modules called: CLRET, ERMSG, FPAR.
C*
C* Data items input: PARBUF.
C*
C* Data items output: ERTEXT, FERR, MDDCB, PARBUF.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM3/MENUSE
COMMON /COM7/PARBUF

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
LOGICAL MENUSE(100)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER ISIZE(2),MDNAM(3),NUMPAR,PARNUM
INTEGER PRDCB(144),PRNAM(3),CVNAM(3)

LOGICAL FERR

```

C Initializations.

```

DATA MDNAM/2HMD,2H00,2H7 /
DATA NUMPAR/3/
DATA PRNAM/2HPR,2HQ0,2H04/,ISIZE/20,0/
DATA CVNAM/2HCV,2H00,2H7 /

```

C Main program section.

```

IF (MENUSE(7)) GOTO 5

```

```

 CALL OPEN(MDDCB,IERR,MDNAM)
 GOTO 7
5 CALL OPEN(MDDCB,IERR,CVNAM)
7 IF (IERR.GE.0) GOTO 10
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) (MDNAM(I),I=1,3)
 CALL ERMSG(FERR)
 GOTO 10101

10 CALL OPEN(PRDCB,IERR,PRNAM)
 IF (IERR.LT.0) CALL CREAT(PRDCB,IERR,PRNAM,ISIZE,4)

20 DO 30 PARNUM = 1,NUMPAR
 CALL FPAR(PARNUM,FERR)
 IF (FERR) GOTO 10101
 CALL WRITF(PRDCB,IERR,PARBUF,40)
30 CONTINUE

10101 CALL CLOSE(MDDCB)
 CALL CLOSE(PRDCB)
 RETURN

9999 FORMAT(5X,"(FP004) Needed menu file MD007 not"
- ," found.")

 END
 ENDS

```

# SUBROUTINE FP006(FERR),19 NOV 83 V1.1

```

C*****
C*
C* Module name: FP006 (FORMAT_PROCESSING_REQUEST_6)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 19 Nov 83
C*
C* Description: Formats the request which is provided to
C* PD006 to guide its actions when it is called upon to
C* perform an IFFT.
C*
C* Calling modules: FREQ.
C*
C* Modules called: CLRET, ERMSG, FPAR.
C*
C* Data items input: PARBUF.
C*
C* Data items output: ERTEXT, FERR, MDDCB, PARBUF.
C*
C*****

```

## C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM3/MENUSE
COMMON /COM7/PARBUF

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
LOGICAL MENUSE(100)
INTEGER PARBUF(40)

```

## C Normal data item definitions.

```

INTEGER ISIZE(2),MDNAM(3),NUMPAR,PARNUM
INTEGER PRDCB(144),PRNAM(3),CVNAM(3)

LOGICAL FERR

```

## C Initializations.

```

DATA MDNAM/2HMD,2H00,2H8 /
DATA NUMPAR/3/
DATA PRNAM/2HPR,2HQ0,2H06/,ISIZE/20,0/
DATA CVNAM/2HCV,2H00,2H7 /

```

## C Main program section.

```

IF (MENUSE(8)) GOTO 5

```

```

 CALL OPEN(MDDCB,IERR,MDNAM)
 GOTO 7
5 CALL OPEN(MDDCB,IERR,CVNAM)
7 IF (IERR.GE.0) GOTO 10
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) (MDNAM(I),I=1,3)
 CALL ERMSG(FERR)
 GOTO 10101

10 CALL OPEN(PRDCB,IERR,PRNAM)
 IF (IERR.LT.0) CALL CREAT(PRDCB,IERR,PRNAM,ISIZE,4)

20 DO 30 PARNUM = 1,NUMPAR
 CALL FPAR(PARNUM,FERR)
 IF (FERR) GOTO 10101
 CALL WRITF(PRDCB,IERR,PARBUF,40)
30 CONTINUE

10101 CALL CLOSE(MDDCB)
 CALL CLOSE(PRDCB)
 RETURN

9999 FORMAT(5X,"(FP006) Needed menu file MD008 not"
- ," found.")

 END
 END$

```

# SUBROUTINE FD001(FERR),8 NOV 83 V1.1

```

C*****
C*
C* Module name: FD001 (FORMAT_DISPLAY_REQUEST_1)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description: Formats the request which is provided to
C* DD001 to guide its actions when it is called upon to
C* accept data from acquisition/processing application
C* programs and display the data on the HP2648A terminal.*
C*
C* Calling modules: FREQ.
C*
C* Modules called: CLRET, ERMSG, FPAR.
C*
C* Data items input: PARBUF.
C*
C* Data items output: ERTEXT, FERR, MDDCB, PARBUF.
C*
C*****

```

## C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM3/MENUSE
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
LOGICAL MENUSE(100)
INTEGER PARBUF(40)

```

## C Normal data item definitions.

```

INTEGER DRDCB(144),DRNAM(3),ISIZE(2)
INTEGER MDNAM(3),NUMPAR,PARNUM,CVNAM(3)

LOGICAL FERR

```

## C Initializations.

```

DATA MDNAM/2HMD,2H01,2H0 /,NUMPAR/4/
DATA DRNAM/2HDR,2HEQ,2H /,ISIZE/20,0/
DATA CVNAM/2HCV,2H01,2H0 /

```

## C Main program section.

```

IF (MENUSE(10)) GOTO 5

```

```

 CALL OPEN(MDDCB,IERR,MDNAM)
 GOTO 7
5 CALL OPEN(MDDCB,IERR,CVNAM)
7 IF (IERR.GE.0) GOTO 10
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999)
 CALL ERMSG(FERR)
 GOTO 10101

10 CALL OPEN(DRDCB,IERR,DRNAM)
 IF (IERR.LT.0) CALL CREAT(DRDCB,IERR,DRNAM,ISIZE,4)

 DO 20 PARNUM = 1,NUMPAR
 CALL FPAR(PARNUM,FERR)
 IF (FERR) GOTO 10101
 CALL WRITF(DRDCB,IERR,PARBUF,40)
20 CONTINUE

10101 CALL CLOSE(MDDCB)
 CALL CLOSE(DRDCB)
 RETURN

9999 FORMAT(5X,"(FD001) Can't find file MD010.")
 END
 END$

```

SUBROUTINE SAREQ(ACQNO,DISNO,FERR,PRONO)  
 -,19 NOV 83, V1.3

```

C*****
C*
C* Module name: SAREQ (SATISFY_REQUEST)
C* Author: Capt John Bengtson
C* Version: 1.3
C* Date: 19 Nov 83
C*
C* Description:
C*
C* SAREQ is responsible for satisfying the user's
C* requests by calling modules which execute application
C* programs for acquiring, processing, and displaying
C* data.
C*
C* Calling module: DSP.
C*
C* Modules called: ADATA, PDATA, DDATA.
C*
C* Data items input: ACQNO, SCRDAT, DISNO, DISREQ, FERR,
C* PRONO, PROREQ.
C*
C* Data items output (changed): ACQIO, ACQNO, DISIO,
C* DISNO, PROIO, PRONO, SARIO.
C*
C*****

```

C Normal data item definitions.

```

 INTEGER ACQNO,DISNO,PRONO(10)
 INTEGER IONUMB(12),SARIO
 INTEGER ACQNAM(3),PRONA1(3),DISNAM(3),BUFFER(5)
 INTEGER PRONA2(3)
 LOGICAL FERR

```

C Initializations.

C Main program section.

C Clear the screen.

```

 WRITE(1,9996) 15530B,15510B,15512B,15452B,
- 62040B,62107B

```

C Get CLASS I/O numbers. Acquisition and display  
 C programs each require a single number; additional  
 C I/O numbers must be acquired for each of the  
 C processing programs to be executed.

```

 DO 10 I = 1,12

```

```

 IONUMB(I) = 0
10 CONTINUE
 SARIO = 0

 ICLAS = 0
 CALL EXEC(20,0,BUFFER,5,IZ,JZ,ICLAS)
 IONUMB(1) = ICLAS
 DO 20 I2 = 1,10
 ICLAS = 0
 CALL EXEC(20,0,BUFFER,5,IZ,JZ,ICLAS)
 IONUMB(I2+1) = ICLAS
 IF (PRONO(I2).EQ.0) GOTO 30
20 CONTINUE
30 DO 40 I = 1,I2+1
 IONUMB(I) = IOR(IONUMB(I),20000B)
 CALL EXEC(21,IONUMB(I),BUFFER,5)
40 CONTINUE
 CALL EXEC(20,0,BUFFER,5,IZ,JZ,SARIO)
 SARIO = IOR(SARIO,20000B)
 CALL EXEC(21,SARIO,BUFFER,5)

```

C Form names of programs to be called. Each name  
C is the concatenation of the process number with  
C a few textual characters.

```

 CALL CODE
 WRITE(ACQNAM,9999) ACQNO
 CALL SCHAR(FERR,1H ,1H0,1,5,ACQNAM)

 CALL CODE
 WRITE(PRONA1,9998) PRONO(1)
 CALL SCHAR(FERR,1H ,1H0,1,5,PRONA1)
 CALL CODE
 WRITE(PRONA2,9998) PRONO(2)
 CALL SCHAR(FERR,1H ,1H0,1,5,PRONA2)

 CALL CODE
 WRITE(DISNAM,9997) DISNO
 CALL SCHAR(FERR,1H ,1H0,1,5,DISNAM)

```

C Schedule programs for execution, passing them  
C any needed I/O numbers.

```

 CALL EXEC(10,ACQNAM,IONUMB(1),IONUMB(2),SARIO)
 IF (PRONO(1).NE.0)
- CALL EXEC(10,PRONA1,IONUMB(2),IONUMB(3),SARIO)
 IF (PRONO(2).NE.0)
- CALL EXEC(10,PRONA2,IONUMB(3),IONUMB(4),SARIO)
 CALL EXEC(10,DISNAM,IONUMB(I2+1),SARIO)

```

C Now wait for completion message from program.



```

 CALL EXEC(21,SARIO,BUFFER,5)

C Release CLASS I/O numbers so that we don't run
C out of them.

 DO 50 I = 1,I2+1
 IONUMB(I) = IOR(100000B,IAND(17777B,IONUMB(I)))
 CALL EXEC(21,IONUMB(I),BUFFER,5)
50 CONTINUE

 SARIO = IOR(100000B,IAND(17777B,SARIO))
 CALL EXEC(21,SARIO,BUFFER,5)

10101 RETURN

9999 FORMAT("AD",I3," ")
9998 FORMAT("PD",I3," ")
9997 FORMAT("DD",I3," ")
9996 FORMAT(10A2)

 END
 END$

```

# SUBROUTINE BORDR(CURMEN,FERR),3 OCT 83, V1.1

```

C*****
C*
C* Module name: BORDR (DRAW_BORDER)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 3 Oct 83
C*
C* Description:
C*
C* BORDR prepares the terminal screen for each
C* menu display. First it clears the screen, then
C* it draws a border around it and enables both
C* alphabetic and graphic display capabilities.
C* If the "Title Page" menu is being displayed
C* (the first one to appear when the system is
C* brought up), some extra lines/boxes and text are
C* written to the screen in addition to the basic
C* border.
C*
C* Calling modules: DMENU, HELP.
C*
C* Modules called: CLRET, CLRTC, CURSR, SESC.
C*
C* Data items input: CONWD2, CURMEN, MAXERT, NOCRLF.
C*
C* Data items output (changed): ERTEXT, FERR, TCONT.
C*
C*****

```

## C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM6/TCONT

COMMON /COM22/CONWD2
COMMON /COM29/MAXERT
COMMON /COM32/NOCRLF

INTEGER ERTEXT(80)
INTEGER TCONT(10)

INTEGER CONWD2
INTEGER MAXERT
INTEGER NOCRLF

```

## C Normal data item definitions.

```

INTEGER CURMEN,LENCHR
LOGICAL FERR

```

C Main program section.  
C Turn off format mode.

```
CALL CLRTC
CALL CODE
WRITE(TCONT,9998)
CALL SESC(FERR,TCONT,1)
TCONT(2) = NOCRLF
CALL EXEC(2,CONWD2,TCONT,2)
```

C Write out basic border-defining instructions to terminal.

```
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999)
CALL SESC(FERR,ERTEXT,45)
ERTEXT(46) = NOCRLF
CALL EXEC(2,CONWD2,ERTEXT,46)
IF (CURMEN.NE.1) GOTO 10101
```

C If this is the "Title Page" menu (CURMEN = 1), write out  
C some extra lines/boxes and text.  
C First draw box.

```
CALL CLRET
CALL CODE
WRITE(ERTEXT,9996)
CALL SESC(FERR,ERTEXT,MAXERT)
CALL EXEC(2,CONWD2,ERTEXT,40)
```

C Position cursor for text.

```
CALL CURSR(2,1,FERR)
```

C Write title page text.

```
WRITE(1,9994)
WRITE(1,9993)
WRITE(1,9992)
```

10101 WRITE(1,9991) 15510B,CURMEN

```
9999 FORMAT("E*d f Z E*d d Z EH EJ E*d a Z E*p a f 0 0 0",
- " 359 719 359 719 0 0 0 Z E*d c Z E*d e Z")
9998 FORMAT("EX")
9996 FORMAT("E*p a f 162 344 553 344 553 259 162 259",
- " 162 344 a 0 244 719 244 Z")
9994 FORMAT(20X,"AFIT Digital Signal Processing System",/)
9993 FORMAT(26X,"Author: Capt John Bengtson")
9992 FORMAT(24X,"Advisor: Professor Gary Lamont")
9991 FORMAT(A2,I3)
```

END  
END\$

SUBROUTINE CLRET  
 -,2 OCT 83, V1.1

```

C*****
C*
C* Module name: CLRET (CLEAR_ERTEXT)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 2 Oct 83
C*
C* Description:
C*
C* CLRET fills the array ERTEXT with spaces.
C*
C* Calling modules: (all that use ERTEXT)
C*
C* Modules called: (none)
C*
C* Data items input: ERTEXT.
C*
C* Data items output: ERTEXT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM29/MAXERT
INTEGER ERTEXT(80)
INTEGER MAXERT

```

C Normal data item definitions.

```

LOGICAL FERR

```

C Main program section.

```

 DO 10 I = 1,MAXERT
 ERTEXT(I) = 2H
10 CONTINUE

10101 RETURN

 END
 END$

```

SUBROUTINE CLRRB  
 -,2 OCT 83, V1.2

```

C*****
C*
C* Module name: CLRRB (CLEAR_RECBUF)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 2 Oct 83
C*
C* Description:
C*
C* CLRRB fills the array RECBUF with spaces.
C*
C* Calling modules: (all that use RECBUF)
C*
C* Modules called: (none)
C*
C* Data items input: MAXREC, RECBUF.
C*
C* Data items output: RECBUF.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM4/RECBUF

COMMON /COM30/MAXREC

INTEGER ERTEXT(80)
INTEGER RECBUF(40)

INTEGER MAXREC

```

C Normal data item definitions.

```

LOGICAL FERR

```

C Main program section.

```

 DO 10 I = 1,MAXREC
10 RECBUF(I) = 2H

10101 RETURN

 END
 END$

```

SUBROUTINE CLRSD,V1.0 26 OCT 83

```
C*****
C*
C* Module name:
C* Author: Capt John Bengtson
C* Version: 1.0
C* Date: 26 Oct 83
C*
C* Description:
C*
C*
C* Calling modules:
C*
C* Modules called:
C*
C* Data items input:
C*
C* Data items output:
C*
C*****
```

C Common block definitions.

COMMON /COM5/SCRDAT

COMMON /COM26/MAXSCR

INTEGER SCRDAT(500)

INTEGER MAXSCR

C Main program section.

```
 DO 10 I = 1,MAXSCR
10 SCRDAT(I) = 0
```

10101 RETURN

END

SUBROUTINE CLRTC  
-,2 OCT 83, V1.1

```
C*****
C*
C* Module name: CLRTC (CLEAR TCONT)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 2 Oct 83
C*
C* Description:
C*
C* CLRTC fills the array TCONT with nulls.
C*
C* Calling modules: (all that use TCONT)
C*
C* Modules called: (none)
C*
C* Data items input: TCONT
C*
C* Data items output: TCONT
C*
C*****
```

C Common block definitions.

```
COMMON /COM1/ERTEXT
COMMON /COM6/TCONT

COMMON /COM31/MAXTCO

INTEGER ERTEXT(80)
INTEGER TCONT(10)

INTEGER MAXTCO
```

C Normal data item definitions.

```
LOGICAL FERR
```

C Main program section.

```
IF (MAXTCO.EQ.10) GOTO 10
FERR = .TRUE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999) MAXTCO
CALL ERMSG(FERR)
STOP
```

```
10 DO 20 I = 1,MAXTCO
 TCONT(I) = 0
```



20 CONTINUE

10101 RETURN

9999 FORMAT("(CLRTC) MAXTCO has been changed from 10 to ",  
- I3,"; stopping execution.")

END

END\$

# SUBROUTINE CURSR(ROW,COLUMN,FERR),20 Oct 83 V1.2

```

C*****
C*
C* Module name: CURSR (POSITION_CURSOR)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 20 Oct 83
C*
C* Description: Positions cursor on the screen of the
C* 2648A terminal.
C*
C* Calling modules: BORDR, ERMSG, HELP, PREC1.
C*
C* Modules called: CLRET, CLRTC, ERMSG, SESC.
C*
C* Data items input: COLUMN, CONWD2, NOCRLF, ROW.
C*
C* Data items output: ERTEXT, FERR, TCONT.
C*
C*****

```

## C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM6/TCONT

COMMON /COM22/CONWD2
COMMON /COM32/NOCRLF

INTEGER ERTEXT(80)
INTEGER TCONT(10)

INTEGER CONWD2
INTEGER NOCRLF

```

## C Normal data item definitions.

```

INTEGER COLUMN,ROW
LOGICAL FERR

```

## C Main program section.

```

IF ((ROW.GE.0).AND.(ROW.LE.23).AND.(COLUMN.GE.0).AND.
- (COLUMN.LE.79)) GOTO 10
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999) 006412B,ROW,COLUMN
CALL ERMSG(FERR)
GOTO 10101

```

```

10 CALL CLRTC

```

HD-A138 232

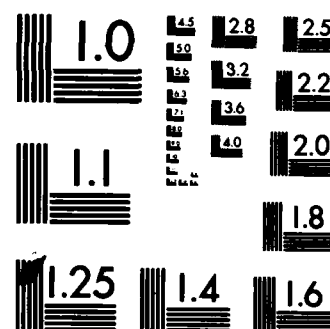
DEVELOPMENT OF A REAL-TIME GENERAL-PURPOSE DIGITAL  
SIGNAL PROCESSING LABO. (U) AIR FORCE INST OF TECH  
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI. J W BENGTON  
DEC 83 AFIT/GCS/EE/83D-3 F/G 9/2

4/4

UNCLASSIFIED

NL

END



```
CALL CODE
WRITE(TCONT,9998) ROW,COLUMN
CALL SESC(FERR,TCONT,6)
TCONT(7) = NOCRLF
CALL EXEC(2,CONWD2,TCONT,7)
```

```
10101 RETURN
```

```
9999 FORMAT(5X,"(CURSR) Row must be between 0 and 23, ",
- A2,5X,"column must be between 0 and 79 (vs. the ",
- I2," ",I2," provided)")
9998 FORMAT("E&a ",I2,"r ",I2," C")
```

```
END
END$
```

# SUBROUTINE ERMSG(FERR),20 Oct 83 V1.2

```

C*****
C*
C* Module name: ERMSG (ERROR_MESSAGE)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 20 Oct 83
C*
C* Description:
C*
C* ERMSG accepts error message text and displays it
C* to the terminal screen in a standard fashion. All
C* error messages are displayed on the last four lines
C* of the screen, with no attention paid to formatting.
C* The user must place carriage returns and line feeds
C* where desired in the midst of the text. Unfortu-
C* nately, using a "/" in a FORMAT statement during
C* the message encoding process doesn't work; it seems
C* to be necessary to place CR/LF in explicitly as an
C* "A2" item (006412B).
C* If the FERR flag is set, signifying a fatal error,
C* then an extra message is printed to tell the user
C* that the program is going to terminate.
C* (See the test program "&FORM" for an example of
C* an attempt to use "/" directly)
C*
C* Calling modules: (all)
C*
C* Modules called: CLRRB, CLRTC, CURSR, SESC.
C*
C* Data items input: ERTEXT, FERR, TCONT.
C*
C* Data items output: ERTEXT, FERR, TCONT.
C*
C*****

```

C Common block definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM4/RECBUF
COMMON /COM6/TCONT

```

```

COMMON /COM22/CONWD2
COMMON /COM29/MAXERT
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF

```

```

INTEGER ERTEXT(80)
INTEGER RECBUF(40)
INTEGER TCONT(10)

```

```
INTEGER CONWD2
INTEGER MAXERT
INTEGER MAXTCO
INTEGER NOCRLF
```

C Normal data item definitions.

```
INTEGER DISCOL,DISROW
LOGICAL FERR
```

C Initializations.

```
DISROW = 21
DISCOL = 0
```

C Main program section.

C Turn off format mode, position cursor, and clear to  
C bottom of screen.

```
CALL CLRTC
CALL CODE
WRITE(TCONT,9999)
CALL SESC(FERR,TCONT,1)
TCONT(2) = NOCRLF
CALL EXEC(2,CONWD2,TCONT,2)
CALL CURSR(DISROW,DISCOL,FERR)
CALL CLRTC
CALL CODE
WRITE(TCONT,9991)
CALL SESC(FERR,TCONT,1)
TCONT(2) = NOCRLF
CALL EXEC(2,CONWD2,TCONT,2)
```

C Write out error message.

```
ERTEXT(MAXERT) = NOCRLF
CALL EXEC(2,CONWD2,ERTEXT,MAXERT)
```

C If fatal error, write out special error message.

C Tell the user to hit "ENTER" to continue.

```
CALL CURSR(23,0,FERR)
CALL CLRRB
CALL CODE
WRITE(RECBUF,9996)
RECBUF(30) = NOCRLF
CALL EXEC(2,CONWD2,RECBUF,30)
```

C Position cursor to top of screen, create a small unprotected field, turn on FORMAT mode, wait for response.

```

DISROW = 1
DISCOL = 1
CALL CODE
WRITE(TCONT,9994) DISROW,DISCOL
LEN = MAXTCO
CALL SESC(FERR,TCONT,LEN)
WRITE(1,9993) (TCONT(I),I=1,LEN)

READ(1,9992) I

10101 RETURN

9999 FORMAT("EX")
9998 FORMAT(8A2)
9997 FORMAT(80A2)
9996 FORMAT(28X,"(Hit ENTER to continue)")
9994 FORMAT("E&a ",I2,"r ",I2,"C ", "E[E]","EW")
9993 FORMAT(10A2)
9992 FORMAT(A2)
9991 FORMAT("EJ")

END
END$

```



SUBROUTINE FPAR(PARNUM,FERR),8 NOV 83 V1.1

```

C*****
C*
C* Module name: FPAR (FILE PARAMETER)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description:
C*
C* FPAR reads MD(NNN) data files and extracts the
C* parameters from non-zero length type-6 records (de-
C* fault values). PARNUM expects that the desired menu
C* data file has been opened using FCB MDDCB.
C*
C* Calling modules: FA000 through FA(NNN), FD000
C* through FD(NNN), FP001 through FP(NNN).
C*
C* Modules called: (none)
C*
C* Data items input: MDDCB, PARNUM
C*
C* Data items output: PARBUF (common area)
C*
C*****

```

C Common block data definitions.

```

COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM7/PARBUF

```

```

INTEGER ERTEXT(80)
INTEGER MDDCB(144)
INTEGER PARBUF(40)

```

C Normal data item definitions.

```

INTEGER FLDLEN,PARCNT,PARNUM,RECTYP

LOGICAL FERR

```

C Main program section.

```

PARCNT = 0
CALL RWNDF(MDDCB)

```

```

C Read until coming to the PARNUMth parameter group
C (counting only those with non-zero length parameters).

```

```

10 DO 20 I = 1,40

```

```

20 PARBUF(I) = 2H

 CALL READF(MDDCB,IERR,PARBUF,40,LEN)
 IF (LEN.GE.0) GOTO 30
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) PARNUM
 CALL ERMSG(FERR)
 GOTO 10101

30 CALL CODE
 READ(PARBUF,9997) RECTYP,FLDLEN
 IF (RECTYP.NE.4) GOTO 10
 IF (FLDLEN.LT.1) GOTO 10

C Found a non-zero length parameter.

 PARCNT = PARCNT+1
 IF (PARCNT.LT.PARNUM) GOTO 10

C If it's the one being looked for, skip the type-5
C record following it and read in the actual default
C value (parameter) desired.

 CALL READF(MDDCB,IERR,PARBUF,40,LEN)
 IF (LEN.GE.0) GOTO 40
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) PARNUM
 CALL ERMSG(FERR)
 GOTO 10101

40 DO 50 I = 1,40
50 PARBUF(I) = 2H

 CALL READF(MDDCB,IERR,PARBUF,40,LEN)
 IF (LEN.GE.0) GOTO 10101
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)

10101 DO 10102 I = 1,39
10102 PARBUF(I) = PARBUF(I+1)
 RETURN

9999 FORMAT(5X,"(FPAR) Can't find requested parameter",
- " in MD(NNN) file ",I3)
9998 FORMAT(5X,"(FPAR) Unexpected EOF in reading MD",

```

- "(NNN) file")  
9997 FORMAT(I1,1X,I3)

END  
END\$

SUBROUTINE NOCR(FERR,MESG,STRLEN)  
-,2 OCT 83, V1.1

```
C*****
C*
C* Module name: NOCR (NO_CARRIAGE_RETURN)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 2 Oct 83
C*
C* Description:
C*
C* NOCR places an underscore character in the
C* user's message string (MESG) just after the char.
C* position designated by STRLEN. STRLEN is incremented
C* by either one or two, to indicate the new length
C* of the string including the underscore.
C* It is very important to note that STRLEN is the
C* number of CHARACTERS in the string, NOT WORDS.
C* The result of doing all this is a message string
C* that will suppress the printing of a CR/LF sequence
C* at the end of the line, if it is printed using
C* an EXEC call to DVR05.
C*
C* Calling modules: PREC5,PREC6.
C*
C* Modules called: (none)
C*
C* Data items input: STRLEN,MESG.
C*
C* Data items output: STRLEN,MESG.
C*
C*****
```

C Common block definitions.

COMMON /COM1/ERTEXT

COMMON /COM32/NOCRLF

INTEGER ERTEXT(80)

INTEGER NOCRLF

C Normal data item definitions.

INTEGER MESG(40)

INTEGER STRLEN,WDADDR

LOGICAL FERR

C Main program section.

```

 IF ((STRLEN.GE.1).AND.(STRLEN.LE.80)) GOTO 10
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) STRLEN
 CALL ERMSG(FERR)
 GOTO 10101

10 WDADDR = (STRLEN-1)/2+1
 IF (MOD(STRLEN,2).EQ.0) GOTO 20
 MMSG(WDADDR) = IAND(MMSG(WDADDR),077400B)
 MMSG(WDADDR+1) = NOCRLF
 STRLEN = STRLEN+3
 GOTO 10101
20 MMSG(WDADDR+1) = NOCRLF
 STRLEN = STRLEN+2

10101 RETURN

9999 FORMAT("(NOCR) Length of string to be processed by",
- " NOCR should be between 1 and 80, not ",I3,".")

 END
 END$

```

# LOGICAL FUNCTION NUMER(NUMCHR),8 NOV 83 V1.1

```

C*****
C*
C* Module name: NUMER
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description:
C*
C* NUMER returns the value TRUE if the characters in
C* PARBUF are numeric; FALSE otherwise.
C*
C* Calling modules: EM001 through EM(NNN)
C*
C* Modules called: (none)
C*
C* Data items input: NUMCHR, PARBUF
C*
C* Data items output: NUMER, PARBUF.
C*
C*****

```

C Common block definitions.

```
COMMON /COM7/PARBUF
```

```
INTEGER PARBUF(40)
```

C Normal data item definitions.

```
INTEGER CHAR,CPOS,WDADDR
```

C Initializations.

```
NUMER = .TRUE.
```

C Main program section.

```

DO 20 CPOS = 1,NUMCHR
 WDADDR = (CPOS-1)/2+1
 IF (MOD(CPOS,2).EQ.1) GOTO 10
 CHAR = IAND(177B,PARBUF(WDADDR))
 IF (((CHAR.GE.60B).AND.(CHAR.LE.71B)).OR.
- (CHAR.EQ.40B).OR.(CHAR.EQ.55B)
- .OR.(CHAR.EQ.56B)) GOTO 20
 NUMER = .FALSE.
 GOTO 2010
10 CHAR = IAND(177B,PARBUF(WDADDR))
 IF (((CHAR.GE.3000B).AND.(CHAR.LE.34400B)).OR.
- (CHAR.EQ.20000B).OR.(CHAR.EQ.27000B)

```

```
- .OR.(CHAR.EQ.26400B)) GOTO 20
 NUMER = .FALSE.
 GOTO 10101
20 CONTINUE
10101 RETURN

 END
 END$
```

# SUBROUTINE PARAM(PARNUM,FERR),V1.1 5 NOV 83

```

C*****
C*
C* Module name: PARAM
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 5 Nov 83
C*
C* Description:
C*
C* PARAM is pulls out a specified parameter/field
C* from the data buffer SCRDAT. Fields are numbered
C* consecutively starting at 1. The first parameter
C* starts at character 1 in the buffer. Each field is
C* separated from the one following it by the US
C* character (000037B). The last field is NOT termi-
C* nated by a US; it is simply followed by whatever
C* was in the buffer before it was filled with a
C* terminal read -- which should be nulls (binary 0).
C* SCRDAT contains the complete set of fields,
C* PARNUM is the number of the parameter/field
C* desired, and PARBUF is passed back to the calling
C* program with just the one parameter desired.
C* Note that the terminal always blank fills (on
C* the right) the last word sent...
C*
C* Calling modules: AOPT, TMENT.
C*
C* Modules called: SCHAR
C*
C* Data items input: MAXSCR,PARNUM,SCRDAT.
C*
C* Data items output: PARBUF.
C*
C*****

```

## C Common block definitions.

```

COMMON /COM5/SCRDAT
COMMON /COM7/PARBUF

COMMON /COM26/MAXSCR

INTEGER SCRDAT(500)
INTEGER PARBUF(40)

INTEGER MAXSCR

```

## C Normal data item definitions.

```

INTEGER FIRSTC, LASTC, LCHAR, LMASK

```



INTEGER PARCNT, PARNUM, PARWD, RCHAR, RMASK, SEPCHR  
INTEGER WDADDR

LOGICAL FERR

DATA LMASK/077400B/, RMASK/000177B/

C Initializations.

PARCNT = 1  
FIRSTC = 1  
LASTC = 1  
WDADDR = 1

C Main program section..

10 LCHAR = IAND(SCRDAT(WDADDR), LMASK)  
RCHAR = IAND(SCRDAT(WDADDR), RMASK)  
IF ((LCHAR.NE.0).AND.(LCHAR.NE.017400B)) GOTO 20  
LASTC = 2\*WDADDR-2  
GOTO 40  
20 IF ((RCHAR.NE.0).AND.(RCHAR.NE.000037B)) GOTO 30  
LASTC = 2\*WDADDR-1  
GOTO 40  
30 WDADDR = WDADDR+1  
IF (WDADDR.LE.MAXSCR) GOTO 10

C Found US or end of last parameter.

40 IF (PARCNT.GE.PARNUM) GOTO 50  
IF(MOD(LASTC,2).EQ.0) WDADDR = WDADDR+1  
WDADDR = WDADDR+1  
FIRSTC = LASTC+2  
LASTC = FIRSTC  
PARCNT = PARCNT+1  
GOTO 10

C Found desired parameter; copy into PARBUF.

50 CALL SCOPY(FIRSTC, LASTC, SCRDAT, 1, PARBUF, FERR)

10101 RETURN

END  
END\$

```

SUBROUTINE SCHAR(FERR,LFROM,LTO,CBEGIN,CEND,MESG)
-,15 OCT 83, V1.2

```

```

C*****
C*
C* Module name: SCHAR (SUBSTITUTE_CHARACTER)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 15 Oct 83
C*
C* Description:
C*
C* SCHAR replaces all occurrences of the character
C* FROM with the character TO in the string MSG.
C* Only the first LEN characters are examined for
C* replacement. Note that LEN is the number of
C* characters, not words.
C* FROM and TO should be specified in the calling
C* routine by assignment statements like:
C*
C* LFROM = LHX (or) LFROM = 054040B
C* LTO = LHY LTO = 054440B
C*
C* to change all "X"s to "Y"s. Note that the character
C* is in the left byte, and a space character is in
C* the right byte.
C*
C* Calling modules: AOPT, DMENU, PREC6, TMENT.
C*
C* Modules called: CLRET, ERMSG.
C*
C* Data items input: CBEGIN, CEND, LFROM, LTO, MSG.
C*
C* Data items output: ERTEXT, MSG.
C*
C*****

```

C Common block definitions.

```
COMMON /COM1/ERTEXT
```

```
INTEGER ERTEXT(80)
```

C Normal data item definitions.

```
INTEGER MSG(80)
```

```
INTEGER CBEGIN,CEND,CPOS,LEN,LCHAR,LFROM,LMASK
```

```
INTEGER LTO,RCHAR,RFROM,RMASK,RTO,WDADDR
```

```
LOGICAL FERR
```

C Initializations.

LMASK = 077400B  
RMASK = 000177B

C Main program section.  
C Make sure that all parameters are within the bounds of  
C arrays dimensioned in this subroutine.

```
 IF((CBEGIN.LE.CEND).AND.(CBEGIN.GE.1).AND.
- (CEND.LE.80)) GOTO 10
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999) CBEGIN,CEND
 CALL ERMSG
 GOTO 10101
```

C Truncate LFROM and LTO and find their right-byte  
C equivalents.

```
10 LFROM = IAND(LFROM,LMASK)
 LTO = IAND(LTO,LMASK)
 RFROM = LFROM/256
 RTO = LTO/256
```

C Make the substitutions.

```
 DO 30 CPOS = CBEGIN,CEND
 WDADDR = (CPOS-1)/2+1
 LCHAR = IAND(MESG(WDADDR),LMASK)
 RCHAR = IAND(MESG(WDADDR),RMASK)
 IF (MOD(CPOS,2).EQ.0) GOTO 20
 IF (LCHAR.EQ.LFROM)
- MESG(WDADDR) = IOR(LTO,RCHAR)
 GOTO 30
20 IF (RCHAR.EQ.RFROM)
- MESG(WDADDR) = IOR(LCHAR,RTO)
30 CONTINUE
```

10101 RETURN

```
9999 FORMAT("(SCHAR) CBEGIN and CEND must be between",
- " 1 and 80 current values: ",I3," ",I3,".")
```

END  
END\$

```

SUBROUTINE SCOPY(IBEGIN,IEND,INBUF,OBEGIN,OBUF,FERR)
- ,8 NOV 83 V1.1

```

```

C*****
C*
C* Module name: SCOPY (STRING COPY)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 8 Nov 83
C*
C* Description:
C*
C* STRCOPY copies a character string from one buffer to
C* another. INBUF is copied to OBUF; the first character
C* copied from INBUF is at character position IBEGIN, the
C* last character copied from INBUF is at character
C* position IEND; the characters are placed in OBUF
C* with the first character going into character position
C* OBEGIN.
C*
C* Calling modules: PARAM
C*
C* Modules called: ERTEXT.
C*
C* Data items input: IBEGIN, IEND, INBUF, OBEGIN, OBUF
C*
C* Data items output: OBUF
C*
C*****

```

C Common block data item definitions.

```

COMMON /COM1/ERTEXT
INTEGER ERTEXT(80)

```

C Normal data item definitions.

```

INTEGER IBEGIN,IEND,INBUF(250),OBEGIN,OBUF(40)
INTEGER IPOS,IWORD,OPOS,OWORD

LOGICAL FERR

```

C Main program section.

```

OPOS = OBEGIN-1
DO 10 I = 1,40
10 OBUF(I) = 2H

```

C Make sure that the input parameters are acceptable.

```

IF (IBEGIN.LE.IEND) GOTO 12

```

```

 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9999)
 CALL ERMSG(FERR)
 GOTO 10101

12 IF ((IBEGIN.GE.1).AND.(IBEGIN.LE.249)) GOTO 14
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9998)
 CALL ERMSG(FERR)
 GOTO 10101

14 IF ((IEND.GE.1).AND.(IEND.LE.249)) GOTO 16
 FERR = .TRUE.
 CALL CLRET
 CALL CODE
 WRITE(ERTEXT,9997)
 CALL ERMSG(FERR)
 GOTO 10101

```

C Copy the string, character by character. There are four C cases to consider, depending upon the location of the C from and to characters in their respective words (left to C left, left to right, right to left, right to right); each C must be dealt with separately. A character's position C within a word is determined by checking whether its character position value is even or odd. If it is even, the C character is in the left byte; if odd, the right byte.

C The OBUF assignments handle, in order: left to left, C left to right, right to left, and right to right.

```

16 DO 50 IPOS = IBEGIN,IEND
 OPOS = OPOS+1
 IWORD = (IPOS-1)/2+1
 OWORD = (OPOS-1)/2+1
 IF (MOD(IPOS,2).EQ.0) GOTO 30
 IF (MOD(OPOS,2).EQ.0) GOTO 20
 OBUF(OWORD) = IOR(IAND(OBUF(OWORD),000177B),
- IAND(INBUF(IWORD),077400B))
 GOTO 50
20 OBUF(OWORD) = IOR(IAND(OBUF(OWORD),077400B),
- IAND(INBUF(IWORD),077400B)/256)
 GOTO 50
30 IF (MOD(OPOS,2).EQ.0) GOTO 40
 OBUF(OWORD) = IOR(IAND(OBUF(OWORD),000177B),
- IAND(INBUF(IWORD),000177B)*256)
 GOTO 50
40 OBUF(OWORD) = IOR(IAND(OBUF(OWORD),077400B),

```

```
 IAND(INBUF(IWORD),000177B))
50 - CONTINUE
10101 RETURN
9999 FORMAT(5X,"(SCOPY) IBEGIN must be less than or equal",
- ," to IEND.")
9998 FORMAT(5X,"(SCOPY) IBEGIN must be between 1 and 249.")
9997 FORMAT(5X,"(SCOPY) IEND must be between 1 and 249.")
 END
 END$
```

SUBROUTINE SESC(FERR,MESG,LENWDS)  
 -,10 OCT 83, V1.2

```

C*****
C*
C* Module name: SESC (SUBSTITUTE_ESCAPE)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 10 Oct 83
C*
C* Description:
C*
C* SESC takes a string composed of printing ASCII
C* characters and replaces all occurrences of capital
C* "E"'s with an ASCII escape character. This allows
C* terminal control sequences to be generated using
C* quoted character strings in FORMAT statements,
C* instead of having to deal with encoding the octal
C* value for an escape into the string.
C*
C* Calling modules: AOPT, BORDR, CURSR, DMENU, DSP,
C* PREC2, PREC5.
C*
C* Modules called: CLRET, ERMSG.
C*
C* Data items input: LENWDS, MESG.
C*
C* Data items output (changed): ERTEXT, FERR, MESG.
C*
C*****

```

C Common block definitions.

```
COMMON /COM1/ERTEXT
```

```
INTEGER ERTEXT(80)
```

C Normal data item definitions.

```
INTEGER MESG(80)
```

```
INTEGER LCAPE,LCHAR,LESC,LMASK,RCAPE,RCHAR
```

```
INTEGER RESC,RMASK,LENWDS
```

```
LOGICAL FERR
```

C Initializations.

```
DATA LCAPE/042400B/,LESC/015400B/,LMASK/077400B/
```

```
DATA RCAPE/000105B/,RESC/000033B/,RMASK/000177B/
```

C Main program section.

C Make sure that LENWDS isn't greater or less than MESG's

C dimension.

```
IF ((LENWDS.GE.1).AND.(LENWDS.LE.80)) GOTO 10
FERR = .TRUE.
CALL CLRET
CALL CODE
WRITE(ERTEXT,9999) LENWDS
CALL ERMSG(FERR)
GOTO 10101
```

C Look for all occurrences of a capital "E", in both the  
C left half and right half (byte) of each word. Must  
C first decompose each word into its left and right  
C bytes.

```
10 DO 30 I = 1,LENWDS
 LCHAR = IAND(MESG(I),LMASK)
 RCHAR = IAND(MESG(I),RMASK)
```

C Skip anything but a capital "E" in left byte.  
C Replace "E" with ASCII escape character and put  
C left and right bytes back together (IOR).

```
IF (LCHAR.NE.LCAPE) GOTO 20
LCHAR = LESC
MESG(I) = IOR(LCHAR,RCHAR)
```

C Skip anything but a capital "E" in right byte.  
C Replace "E" with ASCII escape character and put  
C left and right bytes back together (IOR).

```
20 IF (RCHAR.NE.RCAPE) GOTO 30
 RCHAR = RESC
 MESG(I) = IOR(LCHAR,RCHAR)
```

```
30 CONTINUE
```

```
10101 RETURN
```

```
9999 FORMAT("(SESC) The length of the string to be ",
- "processed by SESC must be between 1 and 80, not ",
- I3,".")
```

```
END
END$
```



BLOCK DATA GCOM,2 OCT 83, V1.1

C Common block definitions.

C First arrays, then simple variables.

```
COMMON /COM1/ERTEXT
COMMON /COM2/MDDCB
COMMON /COM3/MENUSE
COMMON /COM4/RECBUF
COMMON /COM5/SCRDAT
COMMON /COM6/TCONT
COMMON /COM7/PARBUF
```

C Simple variables.

```
COMMON /COM21/CONWD1
COMMON /COM22/CONWD2
COMMON /COM23/CONWD3
COMMON /COM24/COLMAX
COMMON /COM25/COLMIN
COMMON /COM26/MAXSCR
COMMON /COM27/MAXDCB
COMMON /COM28/MAXMEN
COMMON /COM29/MAXERT
COMMON /COM30/MAXREC
COMMON /COM31/MAXTCO
COMMON /COM32/NOCRLF
COMMON /COM33/ROWMAX
COMMON /COM34/ROWMIN
```

C Now define type, set aside space (arrays, then simple).

```
INTEGER ERTEXT(80)
INTEGER MDDCB(144)
LOGICAL MENUSE(100)
INTEGER RECBUF(40)
INTEGER SCRDAT(250)
INTEGER TCONT(10)
INTEGER PARBUF(40)
```

```
INTEGER CONWD1
INTEGER CONWD2
INTEGER CONWD3
INTEGER COLMAX
INTEGER COLMIN
INTEGER MAXSCR
INTEGER MAXDCB
INTEGER MAXMEN
INTEGER MAXERT
INTEGER MAXREC
INTEGER MAXTCO
INTEGER NOCRLF
```

INTEGER ROWMAX  
INTEGER ROWMIN

C Assign permanent values to some.

DATA MENUSE/100\*.FALSE./

DATA CONWD1/000001B/

DATA CONWD2/000001B/

DATA CONWD3/001401B/

DATA COLMAX/80/

DATA COLMIN/1/

DATA MAXSCR/250/

DATA MAXDCB/144/

DATA MAXMEN/100/

DATA MAXERT/80/

DATA MAXREC/40/

DATA MAXTCO/10/

DATA NOCRLF/000137B/

DATA ROWMAX/20/

DATA ROWMIN/1/

END

END\$

# APPENDIX I Application Program Software Listings

PROGRAM AD001(),12 NOV 83 V1.1

```

C*****
C*
C* Module name: AD001 (GENERATE_DATA)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 12 Nov 83
C*
C* Description: Generates sinusoidal data.
C*
C* Calling modules: ADATA.
C*
C* Modules called: CLRRB.
C*
C* Data items input: ACQIO, PROIO, SARIO.
C*
C* Data items output: (sampled data)
C*
C*****

```

C Normal data item definitions.

```

 INTEGER ACQDCB(144),ACQNAM(3),BUFFER(5),RECBUF(40)
 INTEGER TPROIO,PTSBLK,NUMBLK
 INTEGER ACQIO,SARIO

```

```

 REAL DARRAY(4096)

```

C Initializations.

```

 DATA ACQNAM/2HAR,2HEQ,2H /
 DATA PI/3.141593/

```

C Main program section.

C Get Class I/O numbers by calling RMPAR.

```

 CALL RMPAR(BUFFER)
 ACQIO = BUFFER(1)
 TPROIO = BUFFER(2)
 SARIO = BUFFER(3)

```

C Get parameter information from ACQREQ file.

```

 CALL OPEN(ACQDCB,IERR,ACQNAM)
 CALL CLRRB(RECBUF)
 CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
 CALL CODE

```

```

READ(RECBUF,*) PTSBLK
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) NUMBLK
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) SCALE1
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) FS1
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) FSIN1
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) THETA1
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) SCALE2
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) FS2
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) FSIN2
CALL CLRRB(RECBUF)
CALL READF(ACQDCB,IERR,RECBUF,40,LEN)
CALL CODE
READ(RECBUF,*) THETA2

CALL CLOSE(ACQDCB)

```

C Generate the data

```

A = FSIN1/FS1
B = FSIN2/FS2
DO 200 I = 1,PTSBLK
 DARRAY(I) = SCALE1*SIN(2.*PI*I*A+THETA1*PI/180.)
 - +SCALE2*SIN(2.*PI*I*B+THETA2*PI/180.)
200 CONTINUE

```

C Transmit the data

```

CALL EXEC(20,0,DARRAY,514,I,J,TPIOIO)

```

STOP  
END

SUBROUTINE CLRRB(RECBUF),12 NOV 83 V1.0

C Abbreviated form of normal CLRRB; just fills  
Carray with spaces.

INTEGER RECBUF(40)

DO 10 I = 1,40  
RECBUF(I) = 2H

10 CONTINUE

RETURN

END  
END\$

PROGRAM PD004(),11 NOV 83 V1.2

```
C*****
C*
C* Module name: PD004 (PROCESS_DATA_4)
C* Author: Capt John Bengtson
C* Version: 1.2
C* Date: 11 Nov 83
C*
C* Description: Application program for performing
C* Fast Fourier Transform.
C*
C* Calling modules: PDATA.
C*
C* Modules called: FAST.
C*
C* Data items input: ACQIO, DISIO, PROIO, PTSBLK, NUMBLK.
C*
C* Data items output: (transformed data)
C*
C*****
```

C Normal data item definitions.

```
INTEGER BUFFER(5),PRONAM(3),PRODCB(144),TPROIO
INTEGER RECBUF(40),PTSBLK,NUMBLK
INTEGER DISIO,SARIO
```

```
REAL DARRAY(4098)
```

C Initializations.

```
DATA PRONAM/2HPR,2HQ0,2H04/
```

C Main program section.

C Get CLASS I/O numbers from calling program.

```
CALL RMPAR(BUFFER)
TPROIO = BUFFER(1)
DISIO = BUFFER(2)
SARIO = BUFFER(3)
```

```
CALL OPEN(PRODCB,IERR,PRONAM)
IF (IERR.GE.0) GOTO 10
WRITE(1,9999)
CALL EXEC(20,0,BUFFER,5,I,J,SARIO)
GOTO 10101
```

```
10 CALL CLRRB(RECBUF)
CALL READF(PRODCB,IERR,RECBUF,40,LEN)
IF (LEN.GE.0) GOTO 20
```

```

 WRITE(1,9998)
 CALL EXEC(20,0,BUFFER,5,I,J,SARIO)
 GOTO 10101

20 CALL CODE
 READ(RECBUF,*) PTSBLK
 CALL CLRRB(RECBUF)
 CALL READF(PRODCB,IERR,RECBUF,40,LEN)
 IF (LEN.GE.0) GOTO 30
 WRITE(1,9997)
 CALL EXEC(20,0,BUFFER,5,I,J,SARIO)
 GOTO 10101

30 CALL CODE
 READ(RECBUF,*) NUMBLK

C Accept data from data acquisition program.

 CALL EXEC(21,TPIO,DARRAY,514)

C Perform FFT.

 CALL FAST(DARRAY,PTSBLK)

C Send data on to data display program.

 CALL EXEC(20,0,DARRAY,514,I,J,DISIO)

10101 STOP

9999 FORMAT(10X,"(PD004) Can't open file PRQ004")
9998 FORMAT(10X,"(PD004) No points per block parameter",
- " in PRQ004")
9997 FORMAT(10X,"(PD004) No number of points parameter",
- " in PRQ004")
 END

C SUBROUTINE: FAST
C REPLACES THE REAL VECTOR B(K), FOR K=1,2,...,N,
C WITH ITS FINITE DISCRETE FOURIER TRANSFORM
C-----
C
 SUBROUTINE FAST(B, N)
C
C THE DC TERM IS RETURNED IN LOCATION B(1) WITH B(2) SET TO
C THEREAFTER THE JTH HARMONIC IS RETURNED AS A COMPLEX
C NUMBER STORED AS B(2*J+1) + I B(2*J+2).
C THE N/2 HARMONIC IS RETURNED IN B(N+1) WITH B(N+2) SET TO
C HENCE, B MUST BE DIMENSIONED TO SIZE N+2.
C THE SUBROUTINE IS CALLED AS FAST(B,N) WHERE N=2**M AND
C B IS THE REAL ARRAY DESCRIBED ABOVE.
C

```

```

 DIMENSION B(4098)
 COMMON /CONS/ PII, P7, P7TWO, C22, S22, PI2
C
C IW IS A MACHINE DEPENDENT WRITE DEVICE NUMBER
C
C IW = ILMACH(2)
 IW = 1
C
 PII = 4.*ATAN(1.)
 PI8 = PII/8.
 P7 = 1./SQRT(2.)
 P7TWO = 2.*P7
 C22 = COS(PI8)
 S22 = SIN(PI8)
 PI2 = 2.*PII
 DO 10 I=1,15
 M = I
 NT = 2**I
 IF (N.EQ.NT) GO TO 20
10 CONTINUE
 WRITE (IW,9999)
9999 FORMAT (33H N IS NOT A POWER OF TWO FOR FAST)
 STOP
20 N4POW = M/2
C
C DO A RADIX 2 ITERATION FIRST IF ONE IS REQUIRED.
C
 IF (M-N4POW*2) 40, 40, 30
30 NN = 2
 INT = N/NN
 CALL FR2TR(INT, B(1), B(INT+1))
 GO TO 50
40 NN = 1
C
C PERFORM RADIX 4 ITERATIONS.
C
50 IF (N4POW.EQ.0) GO TO 70
 DO 60 IT=1,N4POW
 NN = NN*4
 INT = N/NN
 CALL FR4TR(INT, NN, B(1), B(INT+1), B(2*INT+1), B(3*
* B(1), B(INT+1), B(2*INT+1), B(3*INT+1))
60 CONTINUE
C
C PERFORM IN-PLACE REORDERING.
C
70 CALL FORD1(M, B)
 CALL FORD2(M, B)
 T = B(2)
 B(2) = 0.
 B(N+1) = T
 B(N+2) = 0.

```



```

 DO 80 IT=4,N,2
 B(IT) = -B(IT)
80 CONTINUE
 RETURN
 END

C
C-----
C SUBROUTINE: FR2TR
C RADIX 2 ITERATION SUBROUTINE
C-----
C
 SUBROUTINE FR2TR(INT, B0, B1)
 DIMENSION B0(4098), B1(4098)
 DO 10 K=1,INT
 T = B0(K) + B1(K)
 B1(K) = B0(K) - B1(K)
 B0(K) = T
10 CONTINUE
 RETURN
 END

C
C-----
C SUBROUTINE: FR4TR
C RADIX 4 ITERATION SUBROUTINE
C-----
C
 SUBROUTINE FR4TR(INT, NN, B0, B1, B2, B3, B4, B5, B6,
 DIMENSION L(15), B0(4098), B1(4098), B2(4098), B3(4098)
 DIMENSION B4(4098), B5(4098), B6(4098), B7(4098)
 COMMON /CONS/ PII, P7, P7TWO, C22, S22, PI2
 EQUIVALENCE (L15,L(1)), (L14,L(2)), (L13,L(3)), (L12,L
 * (L11,L(5)), (L10,L(6)), (L9,L(7)), (L8,L(8)), (L7,
 * (L6,L(10)), (L5,L(11)), (L4,L(12)), (L3,L(13)), (L
 * (L1,L(15))

C
C JTHET IS A REVERSED BINARY COUNTER, JR STEPS TWO AT A TIME
C LOCATE THE REAL PARTS OF INTERMEDIATE RESULTS, AND JI LOCA
C THE IMAGINARY PART CORRESPONDING TO JR.
C
 L(1) = NN/4
 DO 40 K=2,15
 IF (L(K-1)-2) 10, 20, 30
10 L(K-1) = 2
20 L(K) = 2
 GO TO 40
30 L(K) = L(K-1)/2
40 CONTINUE

C
 PIOVN = PII/FLOAT(NN)
 JI = 3
 JL = 2
 JR = 2

```

```

C
DO 120 J1=2,L1,2
DO 120 J2=J1,L2,L1
DO 120 J3=J2,L3,L2
DO 120 J4=J3,L4,L3
DO 120 J5=J4,L5,L4
DO 120 J6=J5,L6,L5
DO 120 J7=J6,L7,L6
DO 120 J8=J7,L8,L7
DO 120 J9=J8,L9,L8
DO 120 J10=J9,L10,L9
DO 120 J11=J10,L11,L10
DO 120 J12=J11,L12,L11
DO 120 J13=J12,L13,L12
DO 120 J14=J13,L14,L13
DO 120 JTHET=J14,L15,L14
TH2 = JTHET - 2
IF (TH2) 50, 50, 90
50 DO 60 K=1,INT
 T0 = B0(K) + B2(K)
 T1 = B1(K) + B3(K)
 B2(K) = B0(K) - B2(K)
 B3(K) = B1(K) - B3(K)
 B0(K) = T0 + T1
 B1(K) = T0 - T1
60 CONTINUE
C
IF (NN-4) 120, 120, 70
70 K0 = INT*4 + 1
KL = K0 + INT - 1
DO 80 K=K0,KL
 PR = P7*(B1(K)-B3(K))
 PI = P7*(B1(K)+B3(K))
 B3(K) = B2(K) + PI
 B1(K) = PI - B2(K)
 B2(K) = B0(K) - PR
 B0(K) = B0(K) + PR
80 CONTINUE
GO TO 120
C
90 ARG = TH2*PIOVN
C1 = COS(ARG)
S1 = SIN(ARG)
C2 = C1**2 - S1**2
S2 = C1*S1 + C1*S1
C3 = C1*C2 - S1*S2
S3 = C2*S1 + S2*C1
C
INT4 = INT*4
J0 = JR*INT4 + 1
K0 = JI*INT4 + 1
JLAST = J0 + INT - 1

```

```

DO 100 J=J0,JLAST
 K = K0 + J - J0
 R1 = B1(J)*C1 - B5(K)*S1
 R5 = B1(J)*S1 + B5(K)*C1
 T2 = B2(J)*C2 - B6(K)*S2
 T6 = B2(J)*S2 + B6(K)*C2
 T3 = B3(J)*C3 - B7(K)*S3
 T7 = B3(J)*S3 + B7(K)*C3
 T0 = B0(J) + T2
 T4 = B4(K) + T6
 T2 = B0(J) - T2
 T6 = B4(K) - T6
 T1 = R1 + T3
 T5 = R5 + T7
 T3 = R1 - T3
 T7 = R5 - T7
 B0(J) = T0 + T1
 B7(K) = T4 + T5
 B6(K) = T0 - T1
 B1(J) = T5 - T4
 B2(J) = T2 - T7
 B5(K) = T6 + T3
 B4(K) = T2 + T7
 B3(J) = T3 - T6
100 CONTINUE
C
 JR = JR + 2
 JI = JI - 2
 IF (JI-JL) 110, 110, 120
110 JI = 2*JR - 1
 JL = JR
120 CONTINUE
 RETURN
 END

```

```

C
C-----
C SUBROUTINE: FORD1
C IN-PLACE REORDERING SUBROUTINE
C-----
C

```

```

 SUBROUTINE FORD1(M, B)
 DIMENSION B(4098)
C
 K = 4
 KL = 2
 N = 2**M
 DO 40 J=4,N,2
 IF (K-J) 20, 20, 10
10 T = B(J)
 B(J) = B(K)
 B(K) = T
20 K = K - 2

```

```

 IF (K-KL) 30, 30, 40
30 K = 2*J
 KL = J
40 CONTINUE
 RETURN
 END

```

```

C
C-----
C SUBROUTINE: FORD2
C IN-PLACE REORDERING SUBROUTINE
C-----
C

```

```

 SUBROUTINE FORD2(M, B)
 DIMENSION L(15), B(4098)
 EQUIVALENCE (L15,L(1)), (L14,L(2)), (L13,L(3)), (L12,L
* (L11,L(5)), (L10,L(6)), (L9,L(7)), (L8,L(8)), (L7,
* (L6,L(10)), (L5,L(11)), (L4,L(12)), (L3,L(13)), (L
* (L1,L(15))
 N = 2**M
 L(1) = N
 DO 10 K=2,M
 L(K) = L(K-1)/2
10 CONTINUE
 DO 20 K=M,14
 L(K+1) = 2
20 CONTINUE
 IJ = 2
 DO 40 J1=2,L1,2
 DO 40 J2=J1,L2,L1
 DO 40 J3=J2,L3,L2
 DO 40 J4=J3,L4,L3
 DO 40 J5=J4,L5,L4
 DO 40 J6=J5,L6,L5
 DO 40 J7=J6,L7,L6
 DO 40 J8=J7,L8,L7
 DO 40 J9=J8,L9,L8
 DO 40 J10=J9,L10,L9
 DO 40 J11=J10,L11,L10
 DO 40 J12=J11,L12,L11
 DO 40 J13=J12,L13,L12
 DO 40 J14=J13,L14,L13
 DO 40 JI=J14,L15,L14
 IF (IJ-JI) 30, 40, 40
30 T = B(IJ-1)
 B(IJ-1) = B(JI-1)
 B(JI-1) = T
 T = B(IJ)
 B(IJ) = B(JI)
 B(JI) = T
40 IJ = IJ + 2
 RETURN
 END

```

```

SUBROUTINE CLRRB(RECBUF)
 INTEGER RECBUF(40)
 DO 10 I = 1,40
 RECBUF(I) = 2H
10 CONTINUE
 RETURN
 END

 BLOCK DATA FFTC,17 NOV 83 V1.0

C Common block definitions for PD004 (FFT) program.
 COMMON /CONS/ PII,P7,P7TWO,C22,S22,PI2

 END
 END$

```

PROGRAM PD006(),11 NOV 83 V1.2

```

C*****
C*
C* Module name: PD006 (IFFT)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 11 Nov 83
C*
C* Description: Performs Inverse Fast Fourier Transform.
C*
C* Calling modules: PDATA.
C*
C* Modules called: FSST.
C*
C* Data items input: NUMBLK, PTSBLK, ACQIO, DISIO, SARIO.*
C*
C* Data items output: (inverse transformed data)
C*
C*****

```

C Normal data item definitions.

```

INTEGER BUFFER(5),PRONAM(3),PRODCB(144),TPROIO
INTEGER RECBUF(40),PTSBLK,NUMBLK
INTEGER DISIO,SARIO

```

```

REAL DARRAY(4098)

```

C Initializations.

```

DATA PRONAM/2HPR,2HQ0,2H06/

```

C Main program section.

C Get CLASS I/O numbers from calling program.

```

CALL RMPAR(BUFFER)
TPROIO = BUFFER(1)
DISIO = BUFFER(2)
SARIO = BUFFER(3)

CALL OPEN(PRODCB,IERR,PRONAM)
IF (IERR.GE.0) GOTO 10
WRITE(1,9999)
CALL EXEC(20,0,BUFFER,5,I,J,SARIO)
GOTO 10101

```

```

10 CALL CLRRB(RECBUF)
CALL READF(PRODCB,IERR,RECBUF,40,LEN)
IF (LEN.GE.0) GOTO 20
WRITE(1,9998)

```

```

 CALL EXEC(20,0,BUFFER,5,I,J,SARIO)
 GOTO 10101

20 CALL CODE
 READ(RECBUF,*) PTSBLK
 CALL CLRRB(RECBUF)
 CALL READF(PRODCB,IERR,RECBUF,40,LEN)
 IF (LEN.GE.0) GOTO 30
 WRITE(1,9997)
 CALL EXEC(20,0,BUFFER,5,I,J,SARIO)
 GOTO 10101

30 CALL CODE
 READ(RECBUF,*) NUMBLK

C Accept data from data acquisition program.

 CALL EXEC(21,TPROIO,DARRAY,514)

C Perform IFFT.

 CALL FSST(DARRAY,PTSBLK)

C Send data on to data display program.

 CALL EXEC(20,0,DARRAY,514,I,J,DISIO)

10101 STOP

9999 FORMAT(10X,"(PD006) Can't open file PRQ006")
9998 FORMAT(10X,"(PD006) No points per block parameter",
- " in PRQ006")
9997 FORMAT(10X,"(PD006) No number of points parameter",
- " in PRQ006")

END

C
C-----
C SUBROUTINE: FSST
C FOURIER SYNTHESIS SUBROUTINE
C-----
C
C SUBROUTINE FSST(B, N)
C
C THIS SUBROUTINE SYNTHESIZES THE REAL VECTOR B(K), FOR
C K=1,2,...,N, FROM THE FOURIER COEFFICIENTS STORED IN THE
C B ARRAY OF SIZE N+2. THE DC TERM IS IN B(1) WITH B(2) EQU
C TO 0. THE JTH HARMONIC IS STORED AS B(2*J+1) + I B(2*J+2)
C THE N/2 HARMONIC IS IN B(N+1) WITH B(N+2) EQUAL TO 0.
C THE SUBROUTINE IS CALLED AS FSST(B,N) WHERE N=2**M AND
C B IS THE REAL ARRAY DISCUSSED ABOVE.

```

```

C DIMENSION B(2)
 COMMON /CONST/ PII, P7, P7TWO, C22, S22, PI2
C
C IW IS A MACHINE DEPENDENT WRITE DEVICE NUMBER
C
 IW = 1
C
 PII = 4.*ATAN(1.)
 PI8 = PII/8.
 P7 = 1./SQRT(2.)
 P7TWO = 2.*P7
 C22 = COS(PI8)
 S22 = SIN(PI8)
 PI2 = 2.*PII
 DO 10 I=1,15
 M = I
 NT = 2**I
 IF (N.EQ.NT) GO TO 20
10 CONTINUE
 WRITE (IW,9999)
9999 FORMAT (33H N IS NOT A POWER OF TWO FOR FSST)
 STOP
20 B(2) = B(N+1)
 DO 30 I=4,N,2
 B(I) = -B(I)
30 CONTINUE
C
C SCALE THE INPUT BY N
C
 DO 40 I=1,N
 B(I) = B(I)/FLOAT(N)
40 CONTINUE
 N4POW = M/2
C
C SCRAMBLE THE INPUTS
C
 CALL FORD2(M, B)
 CALL FORD1(M, B)
C
 IF (N4POW.EQ.0) GO TO 60
 NN = 4*N
 DO 50 IT=1,N4POW
 NN = NN/4
 INT = N/NN
 CALL FR4SYN(INT, NN, B(1), B(INT+1), B(2*INT+1), B(3
* B(1), B(INT+1), B(2*INT+1), B(3*INT+1))
50 CONTINUE
C
C DO A RADIX 2 ITERATION IF ONE IS REQUIRED
C
 60 IF (M-N4POW*2) 80, 80, 70

```



```

70 INT = N/2
 CALL FR2TR(INT, B(1), B(INT+1))
80 RETURN
 END

```

```

C
C-----
C SUBROUTINE: FR2TR
C RADIX 2 ITERATION SUBROUTINE
C-----
C

```

```

 SUBROUTINE FR2TR(INT, B0, B1)
 DIMENSION B0(2), B1(2)
 DO 10 K=1,INT
 T = B0(K) + B1(K)
 B1(K) = B0(K) - B1(K)
 B0(K) = T
10 CONTINUE
 RETURN

 END

```

```

C
C-----
C SUBROUTINE: FR4SYN
C RADIX 4 SYNTHESIS
C-----
C

```

```

 SUBROUTINE FR4SYN(INT, NN, B0, B1, B2, B3, B4, B5, B6,
 DIMENSION L(15), B0(2), B1(2), B2(2), B3(2), B4(2), B5
 * B7(2)
 COMMON /CONST/ PII, P7, P7TWO, C22, S22, PI2
 EQUIVALENCE (L15,L(1)), (L14,L(2)), (L13,L(3)), (L12,L
 * (L11,L(5)), (L10,L(6)), (L9,L(7)), (L8,L(8)), (L7,
 * (L6,L(10)), (L5,L(11)), (L4,L(12)), (L3,L(13)), (L
 * (L1,L(15))

```

```

C
 L(1) = NN/4
 DO 40 K=2,15
 IF (L(K-1)-2) 10, 20, 30
10 L(K-1) = 2
20 L(K) = 2
 GO TO 40
30 L(K) = L(K-1)/2
40 CONTINUE

```

```

C
 PIOVN = PII/FLOAT(NN)
 JI = 3
 JL = 2
 JR = 2

C
 DO 120 J1=2,L1,2

```

```

DO 120 J2=J1,L2,L1
DO 120 J3=J2,L3,L2
DO 120 J4=J3,L4,L3
DO 120 J5=J4,L5,L4
DO 120 J6=J5,L6,L5
DO 120 J7=J6,L7,L6
DO 120 J8=J7,L8,L7
DO 120 J9=J8,L9,L8
DO 120 J10=J9,L10,L9
DO 120 J11=J10,L11,L10
DO 120 J12=J11,L12,L11
DO 120 J13=J12,L13,L12
DO 120 J14=J13,L14,L13
DO 120 JTHET=J14,L15,L14
 TH2 = JTHET - 2
 IF (TH2) 50, 50, 90
50 DO 60 K=1,INT
 T0 = B0(K) + B1(K)
 T1 = B0(K) - B1(K)
 T2 = B2(K)*2.0
 T3 = B3(K)*2.0
 B0(K) = T0 + T2
 B2(K) = T0 - T2
 B1(K) = T1 + T3
 B3(K) = T1 - T3
60 CONTINUE
C
 IF (NN-4) 120, 120, 70
70 K0 = INT*4 + 1
 KL = K0 + INT - 1
 DO 80 K=K0,KL
 T2 = B0(K) - B2(K)
 T3 = B1(K) + B3(K)
 B0(K) = (B0(K)+B2(K))*2.0
 B2(K) = (B3(K)-B1(K))*2.0
 B1(K) = (T2+T3)*P7TWO
 B3(K) = (T3-T2)*P7TWO
80 CONTINUE
 GO TO 120
90 ARG = TH2*PIOVN
 C1 = COS(ARG)
 S1 = -SIN(ARG)
 C2 = C1**2 - S1**2
 S2 = C1*S1 + C1*S1
 C3 = C1*C2 - S1*S2
 S3 = C2*S1 + S2*C1
C
 INT4 = INT*4
 J0 = JR*INT4 + 1
 K0 = JI*INT4 + 1
 JLAST = J0 + INT - 1
 DO 100 J=J0,JLAST

```

```

 K = K0 + J - J0
 T0 = B0(J) + B6(K)
 T1 = B7(K) - B1(J)
 T2 = B0(J) - B6(K)
 T3 = B7(K) + B1(J)
 T4 = B2(J) + B4(K)
 T5 = B5(K) - B3(J)
 T6 = B5(K) + B3(J)
 T7 = B4(K) - B2(J)
 B0(J) = T0 + T4
 B4(K) = T1 + T5
 B1(J) = (T2+T6)*C1 - (T3+T7)*S1
 B5(K) = (T2+T6)*S1 + (T3+T7)*C1
 B2(J) = (T0-T4)*C2 - (T1-T5)*S2
 B6(K) = (T0-T4)*S2 + (T1-T5)*C2
 B3(J) = (T2-T6)*C3 - (T3-T7)*S3
 B7(K) = (T2-T6)*S3 + (T3-T7)*C3
100 CONTINUE
 JR = JR + 2
 JI = JI - 2
 IF (JI-JL) 110, 110, 120
110 JI = 2*JR - 1
 JL = JR
120 CONTINUE
 RETURN
 END

```

C

C-----

C SUBROUTINE: FORD1  
C IN-PLACE REORDERING SUBROUTINE

C-----

C

```

SUBROUTINE FORD1(M, B)
 DIMENSION B(2)

```

C

```

 K = 4
 KL = 2
 N = 2**M
 DO 40 J=4,N,2
 IF (K-J) 20, 20, 10
10 T = B(J)
 B(J) = B(K)
 B(K) = T
20 K = K - 2
 IF (K-KL) 30, 30, 40
30 K = 2*J
 KL = J
40 CONTINUE
 RETURN
 END

```

C

C-----

C SUBROUTINE: FORD2  
 C IN-PLACE REORDERING SUBROUTINE

```

C-----
C
 SUBROUTINE FORD2(M, B)
 DIMENSION L(15), B(2)
 EQUIVALENCE (L15,L(1)), (L14,L(2)), (L13,L(3)), (L12,L
* (L11,L(5)), (L10,L(6)), (L9,L(7)), (L8,L(8)), (L7,
* (L6,L(10)), (L5,L(11)), (L4,L(12)), (L3,L(13)), (L
* (L1,L(15))
 N = 2**M
 L(1) = N
 DO 10 K=2,M
 L(K) = L(K-1)/2
10 CONTINUE
 DO 20 K=M,14
 L(K+1) = 2
20 CONTINUE
 IJ = 2
 DO 40 J1=2,L1,2
 DO 40 J2=J1,L2,L1
 DO 40 J3=J2,L3,L2
 DO 40 J4=J3,L4,L3
 DO 40 J5=J4,L5,L4
 DO 40 J6=J5,L6,L5
 DO 40 J7=J6,L7,L6
 DO 40 J8=J7,L8,L7
 DO 40 J9=J8,L9,L8
 DO 40 J10=J9,L10,L9
 DO 40 J11=J10,L11,L10
 DO 40 J12=J11,L12,L11
 DO 40 J13=J12,L13,L12
 DO 40 J14=J13,L14,L13
 DO 40 JI=J14,L15,L14
 IF (IJ-JI) 30, 40, 40
30 T = B(IJ-1)
 B(IJ-1) = B(JI-1)
 B(JI-1) = T
 T = B(IJ)
 B(IJ) = B(JI)
 B(JI) = T
40 IJ = IJ + 2
 RETURN
 END
 SUBROUTINE CLRRB(RECBUF)

 INTEGER RECBUF(40)

 DO 10 I = 1,40
 RECBUF(I) = 2H
10 CONTINUE

```

RETURN

END

BLOCK DATA FFTC,17 NOV 83 V1.0

C Common block definitions for PD006 (IFFT) program.

COMMON /CONST/ P11,P7,P7TWO,C22,S22,PI2

END

END\$

PROGRAM DD001(),14 NOV 83 V1.1

```

C*****
C*
C* Module name: DD001 (DISPLAY_TO_HP2648A)
C* Author: Capt John Bengtson
C* Version: 1.1
C* Date: 14 Nov 83
C*
C* Description: Graphically displays data on the HP2648A
C* terminal.
C*
C* Calling modules: DDATA.
C*
C* Modules called: CLRRB.
C*
C* Data items input: DISIO, PROIO, SARIO.
C*
C* Data items output: (none)
C*
C*****

```

C Normal data item definitions.

```

 INTEGER DISIO,SARIO,DISDCB(144),DISNAM(3),BUFFER(5)
 INTEGER RECBUF(40),PTSBLK,NUMBLK,DTYPE,PART
 REAL DARRAY(4098)

```

C Initializations.

```

 DATA DISNAM/2HDR,2HEQ,2H /
 DATA IOSIZE/514/

```

C Main program section.

C Get CLASS I/O numbers from calling program.

```

 CALL RMPAR(BUFFER)
 DISIO = BUFFER(1)
 SARIO = BUFFER(2)

```

C Open file containing parameters.

```

 CALL OPEN(DISDCB,IERR,DISNAM)
 IF (IERR.GE.0) GOTO 10
 WRITE(1,9998)
 GOTO 10101
10 CALL CLRRB(RECBUF)
 CALL READF(DISDCB,IERR,RECBUF,40,LEN)
 IF (IERR.GE.0) GOTO 20
 WRITE(1,9997)

```

```

 GOTO 10101
20 CALL CODE
 READ(RECBUF,*) PTSBLK
 PTSBLK = MIN0(PTSBLK,IOSIZE)
 CALL CLRRB(RECBUF)
 CALL READF(DISDCB,IERR,RECBUF,40,LEN)
 IF (IERR.GE.0) GOTO 30
 WRITE(1,9996)
 GOTO 10101
30 CALL CODE
 READ(RECBUF,*) NUMBLK
 NUMBLK = 1
 CALL CLRRB(RECBUF)
 CALL READF(DISDCB,IERR,RECBUF,40,LEN)
 IF (IERR.GE.0) GOTO 40
 WRITE(1,9995)
 GOTO 10101
40 CALL CODE
 READ(RECBUF,9999) DTYPE
 IF (DTYPE.EQ.2HC) PTSBLK = PTSBLK/2+1
 CALL CLRRB(RECBUF)
 CALL READF(DISDCB,IERR,RECBUF,40,LEN)
 IF (IERR.GE.0) GOTO 50
 WRITE(1,9994)
 GOTO 10101
50 CALL CODE
 READ(RECBUF,9999) PART

```

C Accept data from processing program.

```
CALL EXEC(21,DISIO,DARRAY,IOSIZE)
```

C Determine YMIN and YMAX.

```

 IF (DTYPE.EQ.2HC) GOTO 60
 YMIN = DARRAY(1)
 YMAX = YMIN
 DO 55 I = 1,PTSBLK
 IF (DARRAY(I).GT.YMAX) YMAX = DARRAY(I)
 IF (DARRAY(I).LT.YMIN) YMIN = DARRAY(I)
55 CONTINUE
 GOTO 100

60 IF (PART.EQ.2HM) GOTO 80
 IF (PART.EQ.2HR) PART = 1
 IF (PART.EQ.2HI) PART = 0
 IF ((PART.NE.0).AND.(PART.NE.1)) PART = 1
 YMIN = DARRAY(PART+2)
 YMAX = DARRAY(PART+2)
 DO 70 I = 1,PTSBLK
 IF (DARRAY(2*I-PART).GT.YMAX)
- YMAX = DARRAY(2*I-PART)

```

```

 IF (DARRAY(2*I-PART).LT.YMIN)
- YMIN = DARRAY(2*I-PART)
70 CONTINUE
 GOTO 100

80 YMIN = SQRT(DARRAY(1)**2+DARRAY(2)**2)
 YMAX = YMIN
 DO 90 I = 1,PTSBLK,2
 VAL = SQRT(DARRAY(I)**2+DARRAY(I+1)**2)
 IF (VAL.GT.YMAX) YMAX = VAL
 IF (VAL.LT.YMIN) YMIN = VAL
90 CONTINUE

```

C Initiate AUTO PLOT.

```

100 WRITE(1,9990) 15452B
 YINTER = (YMAX-YMIN)/10.
 YTIC = (YMAX-YMIN)/20.
 WRITE(1,9993) 15452B,PTSBLK,YMIN,YMAX
- ,YINTER,YTIC,PTSBLK

```

C Send data to the terminal.

```

 IF (DTYPE.EQ.2HC) GOTO 105
 DO 103 I = 1,PTSBLK
 WRITE(1,9992) I,DARRAY(I)
103 CONTINUE
 GOTO 140

105 IF (PART.EQ.2HM) GOTO 120
 DO 110 I = 1,PTSBLK
 WRITE(1,9992) I,DARRAY(2*I-PART)
110 CONTINUE
 GOTO 140

120 DO 130 I = 1,PTSBLK
 VAL = SQRT(DARRAY(2*I-1)**2+DARRAY(2*I)**2)
 WRITE(1,9992) I,VAL
130 CONTINUE

```

C Turn off AUTO PLOT.

```

140 WRITE(1,9989) 15452B
 WRITE(1,9988) 15452B,15452B
 WRITE(1,9987) 15452B
 WRITE(1,9986) 15452B

```

C Wait for user to respond.

```

 CALL EXEC(1,1,I,1)

```

C Send termination message to SAREQ.



CALL EXEC(20,0,BUFFER,5,I,J,SARIO)

10101 STOP

```
9999 FORMAT(A2)
9998 FORMAT(10X,"(DD001) Can't open file DREQ")
9997 FORMAT(10X,"(DD001) No points per block parameter",
- " found in file DREQ")
9996 FORMAT(10X,"(DD001) No number of blocks parameter",
- " found in file DREQ")
9995 FORMAT(10X,"(DD001) No data type parameter found",
- " in file DREQ")
9994 FORMAT(10X,"(DD001) No data part parameter found",
- " in file DREQ")
9993 FORMAT(A2,"a d 2h 1i 2j 1k 1l ",I4,"m ",E12.4,
- "n ",E12.4,"o 10p 1q ",E12.4,"r ",E12.4,"s ",
- I4,"u 1v 0w c A")
9992 FORMAT(I4,2X,E12.4)
9990 FORMAT(A2,"d a f z")
9989 FORMAT(A2,"m 2m 5Q")
9988 FORMAT(A2,"d 359,15o z",2X,A2,"p a c z")
9987 FORMAT(A2,"d S","(Hit ENTER to Continue)")
9986 FORMAT(A2,"d t z")
```

END

SUBROUTINE CLRRB(RECBUF)

INTEGER RECBUF(40)

DO 10 I = 1,40

RECBUF(I) = 2H

10 CONTINUE

RETURN

END

END\$

## APPENDIX J Test Plan

This test plan is divided up into a series of blocks, with each block covering the testing of one or more (if related) software modules. Each block has a series of consecutively-numbered tests. Each test specifies the actions to be performed and the expected results of those actions. The date on which the test/block of tests was passed is given at the end of each test case/block.

In addition to the "white box" tests given here, each completed test date signifies completion of the "black box" tests presented in Chapter 3.

The test cases are presented in essentially a top-down order, according to where the concerned modules lie in the system's hierarchy tree (this is the same order in which the modules are listed in Appendices H and I).

Unless stated otherwise, it is assumed that all modules subordinate to the one/those being tested are stubbed.

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): DSP

(1) FERR = TRUE from GREQ

=>

Screen is cleared, terminal is reset, execution stops

(2) FERR = TRUE from SAREQ

=>

Screen is cleared, terminal is reset, execution stops

(3) FEXIT = TRUE from GREQ

=>

Screen is cleared, terminal is reset, execution stops

(4) FEXIT = FALSE and FERR = FALSE from GREQ  
=>  
SAREQ is called; on return, if not FERR, back to GREQ  
Date passed: 20 Oct 83

\*\*\*\*\*

Module(s): GREQ  
(1) FERR = TRUE or FEXIT = TRUE from GOPT  
=>  
Module is exited; otherwise, execution continues with  
calling EOPT  
(2) FERR = TRUE or FEXIT = TRUE from EOPT  
=>  
Module is exited; otherwise, check FGEDIT  
(3) FGEDIT = FALSE from EOPT  
=>  
Go back and call GOPT again  
(4) FERR = FALSE, FEXIT = FALSE, FGEDIT = TRUE, FESE = TRUE  
=>  
Call FREQ to format user's request  
Date passed: 15 Oct 83

\*\*\*\*\*

Module(s): GOPT  
(1) FERR = TRUE from DMENU  
=>  
GOPT is exited  
(2) FERR = FALSE from DMENU  
=>  
AOPT is called  
(3) FERR = FALSE or FREDIS = FALSE from AOPT  
=>  
GOPT is exited  
else go back and call DMENU again  
Date passed: 27 Oct 83

\*\*\*\*\*

Module(s): DMENU and PREC1 through PREC6  
(1) Nothing but type-0 records in menu file  
=>  
Write appropriate error message and exit from DMENU  
(2) Type-0 records followed by non type-1 record  
=>  
Error message and exit from DMENU  
(3) Missing record of any one type in record group  
=>  
Record-specific error message and exit  
(4) Records out of order  
=>  
Error message and exit

- (5) Row or column numbers out of range for type-1 record  
=>  
Error message and exit
- (6) Attribute character <> "I", "N", or " " for type-2 record  
=>  
Error message and exit
- (7) Response field length in PREC4 < 0 or > 78  
=>  
Error message and exit
- (8) No errors in above items  
=>

Menu is displayed to terminal exactly as defined  
Date passed: 29 Oct 83 and 1 Nov 83

\*\*\*\*\*

Module(s): TMENT

- (1) Non-existent current menu entry in MENUT file  
=>  
Error message and exit
- (2) No errors  
=>

New CURMEN value is provided as requested  
Date passed: 4 Nov 83

\*\*\*\*\*

Module(s): AOPT

- (1) Invalid function key is pressed  
=>  
No action; menu remains static
- (2) Valid function key is pressed  
=>  
Appropriate flag is set
- (3) Request to save current values as new defaults  
=>  
All current values are copied into proper MD(NNN) file
- (4) Immediate request (function key)  
=>

SIOPT is called with proper flag settings  
Date passed: 27 Oct 83

\*\*\*\*\*

Module(s): SIOPT

- (1) FBACK = TRUE  
=>  
Call TMENT and exit
- (2) FDIREC = TRUE  
=>  
Call DIREC and exit

(3) FHELP = TRUE  
=>  
Call HELP and exit  
(4) FSVREQ = TRUE  
=>  
Call SVREQ and exit  
Date passed: 15 Oct 83

\*-\*-\*-\*-\*

Module(s): HELP

(1) If FERR is set by any subroutines  
=>  
Error message and exit  
(2) If help file contains multiple pages  
=>  
ENTER and BACKUP keys allow user to move back and forth  
between pages at will  
(3) EXIT key hit  
=>  
User is taken back to originating menu  
Date passed: 15 Oct 83

\*-\*-\*-\*-\*

Module(s): EOPT

(1) If FERR is set by any subroutine  
=>  
Exit EOPT  
(2) CURMEN between 1 and 10  
=>  
Call exactly one of EM001 through EM010 corresponding to  
the value of CURMEN  
Date passed: 2 Oct 83

\*-\*-\*-\*-\*

Module(s): EM001 through EM010

(1) If invalid CHOICE value  
=>  
Error message, set FGEDIT to FALSE, and exit edit module  
Date passed: 31 Oct 83, 8 Nov 83, and 19 Nov 83

\*-\*-\*-\*-\*

Module(s): FREQ

(1) If invalid ACQNO, DISNO, PRONO values  
=>  
Error message (fatal) and exit  
(2) If valid ACQNO, DISNO, PRONO values  
=>  
Call appropriate FA(NNN), FP(NNN), FD(NNN) module(s)  
Date passed: 31 Oct 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): FA001

(1) Menu data needed for acquisition request is from menu data file (MD(NNN)) that has been used during current terminal session

=>

Data values are taken from CV(NNN) file

Date passed: 8 Nov 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): FP004 and FP006

(1) Menu data for processing request is from menu data file that has been used during current terminal session

=>

Data values are taken from CV(NNN) file

Date passed: 19 Nov 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): FD001

(1) Menu data for display request is from menu data file that has been used during current terminal session

=>

Data values are taken from CV(NNN) file

Date passed: 8 Nov 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): SAREQ

(1) ACQNO = N

=>

AD(N) executed

(2) PRONO(1) = M, PRONO(2) = N, ...

=>

PD(M) executed as process #1, PD(N) executed as process #2, etc.

(3) DISNO = N

=>

DD(N) executed

Date passed: 19 Nov 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): BORDR

(1) If CURMEN = 1

=>

Write out special border and text at top of screen

Date passed: 3 Oct 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): CLRET, CLRRB, CLRSD, CLRTC

(1) If change in MAX(XXX)

=>

Error message and stop (may be disabled/deleted)

Date passed: 2 Oct 83 and 26 Oct 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): CURSR

(1) If ROW < 0 or ROW > 23 or COLUMN < 0 or COLUMN > 79

=>

Error message and exit

else

Position cursor as requested

Date passed: 20 Oct 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): ERMSG

(1) Screen with protected fields, etc. defined when ERMSG  
is called

=>

Screen area for printing error message is cleared by  
ERMSG before printing begins

Date passed: 20 Oct 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): FPAR

(1) Requested parameter doesn't exist in the file

=>

Error message and exit

(2) Requested parameter is shorter than descriptor says

=>

Blank fill on the right

Date passed: 8 Nov 83

\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*--\*

Module(s): NOCR

(1) STRLEN an even value

=>

NOCR value placed in following word

(2) STRLEN an odd value

=>

NOCR value placed in low-byte position of current word

Date passed: 2 Oct 83

\*-\*-\*-\*-\*

Module(s): NUMER

(1) NUMCHR an even or odd number

=>

Correct logical value is returned (i.e., character isolation is handled properly)

Date passed: 8 Nov 83

\*-\*-\*-\*-\*

Module(s): PARAM

(1) Only one parameter present in SCRDAT

=>

Single parameter correctly returned (absence of US character causes no problems)

Date passed: 5 Nov 83

\*-\*-\*-\*-\*

Module(s): SCHAR

(1) Substitute starting in odd-byte position

=>

Translation handled correctly

Date passed: 15 Oct 83

\*-\*-\*-\*-\*

Module(s): SCOPY

(1) Copy from odd-byte position to even-byte position

=>

Character translation handled correctly

(2) Copy from even-byte position to odd-byte position

=>

Character translation handled correctly

Date passed: 8 Nov 83

\*-\*-\*-\*-\*

Module(s): SESC

(1) "E" located in left and right bytes of words

=>

Translation correctly handled

(2) No "E"'s found in string (or empty string)

=>

No action taken

Date passed: 10 Oct 83



[illegible]

\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*

### Vita

Captain John W. Bengtson was born May 19, 1956 in Bakersfield, California. He was raised in California's San Joaquin Valley, attending Lemoore High School, West Hills College, and California State University, Fresno (BA in mathematics) on his way to an ROTC-gained commission as a Second Lieutenant in the Air Force. Following his May of 1978 commission, he was assigned to the Air Force Data Services Center (HQ USAF). There he served as leader of the team responsible for AF/LE computer support (receiving the Air Force Meritorious Service Medal for his performance) and as a White House Social Aide. He entered AFIT in June of 1982 to pursue a Master of Science in Computer Systems.

Permanent address: 21810 Geneva Ave.  
Lemoore, CA 93245

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

|                                                                                                                                                                                                                                  |                                                 |                                                                                                   |                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|---------------------------------------------------------------------------------------------------|-------------------------------|
| 1. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED                                                                                                                                                                                |                                                 | 1d. RESTRICTIVE MARKINGS                                                                          |                               |
| 2a. SECURITY CLASSIFICATION AUTHORITY                                                                                                                                                                                            |                                                 | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release;<br>distribution unlimited. |                               |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE                                                                                                                                                                                        |                                                 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)                                                       |                               |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>AFIT/GCS/EE/83D-3                                                                                                                                                                 |                                                 | 7a. NAME OF MONITORING ORGANIZATION                                                               |                               |
| 6a. NAME OF PERFORMING ORGANIZATION<br>School of Engineering                                                                                                                                                                     | 6b. OFFICE SYMBOL<br>(If applicable)<br>AFIT/EN | 7b. ADDRESS (City, State and ZIP Code)                                                            |                               |
| 6c. ADDRESS (City, State and ZIP Code)<br>Air Force Institute of Technology<br>Wright-Patterson AFB, Ohio 45433                                                                                                                  |                                                 | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER                                                   |                               |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION                                                                                                                                                                                      | 8b. OFFICE SYMBOL<br>(If applicable)            | 10. SOURCE OF FUNDING NOS.                                                                        |                               |
| 8c. ADDRESS (City, State and ZIP Code)                                                                                                                                                                                           |                                                 | PROGRAM ELEMENT NO.                                                                               | PROJECT NO.                   |
| 11. TITLE (Include Security Classification)<br>See box 19 (Unclassified)                                                                                                                                                         |                                                 | TASK NO.                                                                                          | WORK UNIT NO.                 |
| 12. PERSONAL AUTHOR(S)<br>John W. Bengtson, B.A., Capt, USAF                                                                                                                                                                     |                                                 |                                                                                                   |                               |
| 13a. TYPE OF REPORT<br>MS Thesis                                                                                                                                                                                                 | 13b. TIME COVERED<br>FROM _____ TO _____        | 14. DATE OF REPORT (Yr., Mo., Day)<br>1983 December                                               | 15. PAGE COUNT<br>345         |
| 16. SUPPLEMENTARY NOTATION<br>Approved for public release: IAW AFR 190-17.<br>LYNN E. WOLLAVER 1 Feb 84<br>Dean for Research and Professional Development<br>Wright-Patterson AFB OH 45433                                       |                                                 |                                                                                                   |                               |
| 17. COSATI CODES                                                                                                                                                                                                                 |                                                 | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)                 |                               |
| FIELD                                                                                                                                                                                                                            | GROUP                                           | SUB. GR.                                                                                          |                               |
| 9                                                                                                                                                                                                                                | 2                                               | Computers, Digital Computers, Signal Processing, Real Time                                        |                               |
| 19. ABSTRACT (Continue on reverse if necessary and identify by block number)<br><br>Title: DEVELOPMENT OF A REAL-TIME, GENERAL-PURPOSE DIGITAL SIGNAL PROCESSING LABORATORY SYSTEM<br><br>Thesis Chairman: Professor Gary Lamont |                                                 |                                                                                                   |                               |
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>                                                |                                                 | 21. ABSTRACT SECURITY CLASSIFICATION<br>UNCLASSIFIED                                              |                               |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Professor Gary Lamont                                                                                                                                                                     |                                                 | 22b. TELEPHONE NUMBER<br>(Include Area Code)<br>(513) 255-2057                                    | 22c. OFFICE SYMBOL<br>AFIT/EN |

Abstract (Block 19)

This investigation resulted in the design and implementation of software to support a real-time, general-purpose digital signal processing (DSP) system. The major design aims for the system were that it: be easy to use, support a wide variety of DSP functions, and be capable of real-time processing. All work was performed using an HP21MX computer running under the RTE-III operating system.

The system's analysis and design were accomplished using Structured Analysis and Structured Design techniques. Their results -- both logical and physical system designs -- are presented via Data Flow Diagrams and Structure Charts respectively. The hardware environment was also analyzed to ensure its suitability.

The resulting system consists of two main components: a User Interface, and a collection of DSP application programs. The User Interface is menu driven and allows the system to be used by those with little or no prior computer experience. The User Interface gathers user requests and presents them to an arbitrary number of concurrently executing application programs for satisfaction. All of the major system components were successfully implemented with the exception of real-time data sampling support via analog to digital converter. With the addition of an HP21MX co-processor, the developed system should be capable of supporting the full range of DSP activities envisioned.

**END**

**FILMED**

**384**

**DTIC**